Studies on Logical Design Theory and Physical Storage Structures for Relational Databases

Katsumi Tanaka

January 1981

Studies on Logical Design Theory and Physical Storage Structures for Relational Databases

Katsumi Tanaka

January 1981

Abstract

This thesis includes the following five topics in logical database design theory and physical storage structures of relational databases: (1) properties of embedded multivalued dependencies (EMVDs), (2) preservation of data dependencies, (3) semantic aspects of data dependencies, (4) organization of Quasi-Consecutive Retrieval (QCR) files and (5) organization of Relational Inverted Structure (RIS) files. The topics (1), (2) and (3) are mainly concerned with specification and maintenance of an important class of integrity constraints, called data dependencies, in relational databases. The topics (4) and (5) are concerned with the physical data organization of relational database systems towards rapid retrieval and reduction of required storage space.

In Chapter 1, the backgrounds, objectives, motivations and the outline of the thesis are described. Especially, the relationships among the above five topics are explained. In Chapter 2, basic concepts of relational databases are summarized.

Chapter 3 is concerned with the properties of EMVDs. Several new inference rules for EMVDs are shown, which are useful to handle several problems of Fagin's multivalued dependencies. Especially, a basic theorem about the interactions between a multivalued dependency and an EMVD is shown. This theorem provides a necessary and sufficient condition for a multivalued dependency, which holds in a certain set of attributes, to hold in its superset of attributes.

Chapter 4 is concerned with the preservation of data dependencies by a set of relation schemes. When we allow each relation to be updated independently from others, we mainly

discuss what data dependencies can be enforced by a set of relation schemes, on each of which several dependency constraints are given. The notion of the Dependency Preserving Normal Form (DPNF) is introduced. Under the assumtion that each relation scheme is in DPNF, a necessary and sufficient condition for the set of relation schemes to enforce a data dependency (functional, embedded multivalued or embedded join dependency) is provided.

Chapter 5 is concerned with the semantic aspects of functional and multivalued dependencies. Several differences between a conceptual design and a logical design by data dependencies are clarified. Problems of multivalued dependencies are also discussed.

Chapter 6 is concerned with the organization of QCR files. The QCR file offers normally less redundancy than Ghosh's consecutive retrieval files in storing relations or secondary indexes. From a viewpoint of a graph theory, a necessary and sufficient condition for the existence of a QCR file without redundancy is provided. Furthermore, a heuristic computer algorithm to organize a QCR file with less redundancy is also shown.

Chapter 7 is concerned with the organization of RIS files. The RIS file is a kind of combination of multiple secondary indexes. By RIS files and the notion of 'pseudo' operations, efficient processing algorithms for relational operations are provided. The 'pseudo' operation is to process relational operations not on actual data, but on hashed values of data. It is useful to decrease the number of accesses to relations and to achieve a rapid retrieval.

Contents

Abbreviations and Symbols

$A_i \sim A_j$: Attributes A_i and A_j are domain-related.

A,B,C,...: Attributes. Possibly with subscripts.

BCNF: Boyce-Codd Normal Form.

CEED: Conceptual element-element dependency.

CESD: Conceptual element-set dependency.

CR file: Consecutive retrieval file.

$DOM(A_i)$ or D_i: A domain of attribute A_i.

$d(q_j)$: The number of edges each of which connects the non-redundant vertex q_j and a redundant vertex in G^c of M'.

D(Z): A dependency diagram of a set Z of attributes.

EJD: Embedded join dependency.

$E(q_j)$: A summation of $|R(q_j) \cap R(q_i)|$ for $1 \leq i \leq n$, $j \neq i$.

EMVD: Embedded multivalued dependency.

FD: Functional dependency.

FMVD: Full multivalued dependency.

G: A query graph of a given RQ matrix.

G^c: A complement graph of a graph G.

G^{c*}: A partially ordered graph of a complement graph G^c.

h(x): A hashed value of x.

h(A): A hashed value of a value of attribute A.

JD: Join dependency.

k: A buffer size of a QCR file.

M: A record-query incidence matrix.

M': A record-query incidence matrix with redundant queries.

MVD: Multivalued dependency.

$N_j(q)$: The j-th neighbourhood of query q.

QCR file: Quasi-consecutive retrieval file.

q_i: A query.

R: A relation scheme or a set of records (in Chapter 6). Possibly with subscripts.

r_i: A relation on R_i or a record (in Chapter 6).

RQ matrix: A record-query incidence matrix.

$Q(r_i)$: A set of queries to which record r_i is pertinent.

$R(q_i)$: A set of records which are pertinent to query q_i.

RIS: Relational inverted structure.

$RIS(A_i)$: A RIS file for a set of attributes which are domain-related to attribute A_i.

r[X]: A projection of relation r on a set X of attributes.

r[x,Y]: A projection of a subrelation on Y of relation r in which every tuple's X-value is x.

$\{r_1,\ldots,r_n\}$: A relational database (except Chapter 6).

$\{R_1,\ldots,R_n\}$: A relational database scheme (except Chapter 6).

r[A='c']: a selection of relation r on attribute A.

$r_1[A \div B]r_2$: A division of r_1 on A by r_2 on B.

$\overset{n}{\underset{i=1}{*}} r_i$: A natural join of relations $r_1,\ldots,r_n$.

$*[R_1,\ldots,R_n]$: A join dependency or an embedded join dependency.

SAT(C): A set of all the relations on a relation scheme that satisfy a set C of constraints.

t: A tuple or a mapping from a set of attributes to a domain. Possibly with subscripts.

TID: Tuple identification code.

$t(A_i)$: A value of attribute A_i of a tuple t.

$t[A_i]$: A restriction of a mapping t onto A_i.

X -> Y: A functional dependency. $X \rightarrow Y$ is also used.

X ->-> Y: A multivalued dependency. $X \twoheadrightarrow Y$ is also used.

X ->-> Y|Z: An embedded multivalued dependency. $X \twoheadrightarrow Y|Z$ is also used.

$X \rightarrow\!\!\!\rightarrow Y_1|\ldots|Y_k$: $\{Y_1,\ldots,Y_k\}$ is a dependency basis of a set X of attributes.

$X \Rightarrow\!\!\!\Rightarrow Y$: A full multivalued dependency.

$X_1|\ldots|X_k \Rightarrow\!\!\!\Rightarrow Y$: $\{X_1,\ldots,X_k\}$ is a left dependency basis of a set Y of attributes.

$\langle X \rangle$: A node corresponding to a set X of attributes in a dependency diagram.

$(\langle X_i \rangle, \langle X_j \rangle)$: An edge connecting vertices $\langle X_i \rangle$ and $\langle X_j \rangle$ in a dependency diagram.

$X \Rightarrow Y$: A conceptual element-element dependency.

$X \Rightarrow\!\!\!\Rightarrow Y$: A conceptual element-set dependency.

...X,Y,Z: Sets of attributes. Possibly with subscripts.

z_j: A number of zeros within the retrieval area of the j-th column in an RQ matrix.

1NF: First normal form.

4NF: Fourth normal form.

2^R: A power set of a set R.

ϕ: An empty set.

CHAPTER 1 INTRODUCTION

1.1 Backgrounds

Database research is now one of the most active research areas in computer science. In this field, thousands of research papers have been published for these past ten years [KAMB81]. In many computer systems, several kinds of database management systems have become available as fundamental software utilities. Rapid progresses in hardware technology, especially in storage media such as magnetic disks, magnetic bubbles and mass storage systems, enabled us to store and retrieve a large amount of data in computer systems. This stimulates and enlarges the areas of database applications not only to business oriented data processing and bibliographic information retrieval, but also to other application areas.

The major benefits of the database approach are (1) data sharing and elimination of redundant data, (2) integrated control of data which is useful to maintain consistency and integrity of data and (3) data independence from both application programs and physical data organizations, which is useful to enlarge the life cycle of programs and data. Most theoretical and practical researches on databases have been concerned with how to realize these issues.

In 1970, E. F. Codd introduced the relational data model for databases [CODD7006]. In the relational data model, data values are arranged into flat tables called relations. This data model is widely spread as a basis of database theory because of its simplicity and its mathematical concept 'relations'. In practical aspects, it is also useful to achive a high degree of data independence by providing a simple view of

data.

In a series of Codd's succeeding papers, he also introduced two important concepts. One is the notion of functional dependencies, which are integrity constraints of data [CODD7105F]. The contents of a database is time-varying since several update operations are performed on data. However, a subset of all the possible databases is usually of interest for some enterprise. Integrity constraints are basically predicates on data, which are used to specify a set of possible meaningful databases. Codd also provided a database design method, called normalization, which is useful to maintaining given functional dependencies and to reduce the redundancy of data. The other is the concept of relational completeness of database languages based on relational calculus and relational algebra [CODD7105R]. The relational algebra and the relational calculus are important in both practical and theoretical aspects. Practically, they offer very flexible and powerful data manipulation capabilities. The concept of completeness is useful to consider the capability of a database language from a theoretical view.

Since the introduction of Codd's relational data model, the following two research areas have become to be important. One is a research on the logical design of a relational database and data dependency theory. Several integrity constraints involving Codd's functional dependencies have been studied in order to choose a good logical design [BEERB7809] [RISS7809]. The first half of this thesis is devoted to the study in this research area. The other is concerned with how to implement a relational database system physically in a reasonable performance. Although the relational data model offers a flexible data manipulation facility, there are several problems to be solved in its physical implementation. Especially, the research on its

physical data organization is important in order to have a good performance. The latter half of this thesis is concerned with this topic.

1.2 Objectives of the Thesis

In designing and implementing a relational database system, the following are important criteria of the performance: (1) retrieval time, (2) reduired storage space and (3) ease of update maintenance. In this thesis, we discuss several problems of (1), (2) and (3) from two aspects: data dependency theory and physical data organization.

In order to achieve a rapid retrieval, it is important to consider how to organize and store data physically in computer systems. In the relational data model, data are logically arranged into relations. Relations are, however, not necessarily stored in the forms of tables. Furthermore, in most database systems, additional data is also stored in order to achieve rapid retrieval, such as secondary indexes and links. The physical data organization of relations and these additional data also depends on allowable retrieval operations. As for rapid retrieval, the problems discussed in this thesis are as follows:

(a) How to increase the locality of data: Even though several additional data is used, it will take a lot of time to retrieve actual data from relations if the resultant data is stored dispersedly. Therefore, it is useful to consider how to organize data in a way that the resultant data for every query is clustered as much as possible.

(b) How to process relational operations efficiently: If a

relational database systems store only relations in forms of tables, rapid retrieval will not be possible for several relational operations, such as selection, join and division [CODD7105R]. Secondary indexes are useful to process selection operations, and links are useful to process join operations. The division operation is, however, not supported efficiently by either secondary indexes or links. Division operation is an important operation to formulate a query concenred with universal quantifiers. Furthermore, the coexistence of secondary indexes and links will lead to the necessity of complicated maintenance.

As for (a), the notion of 'quasi-consecutive retrieval files' is introduced in this thesis. It is a generalization of the consecutive retrieval files introduced by Ghosh [GHOS7209]. Ghosh's consecutive retrieval files are useful to increase the locality of data since every resultant data for any query are consecutively placed in the files. Several restrictions on consecutive retrieval files are, however, too strict and so not practical. Therefore, we relax those restrictions and generalize it in this thesis. As for (b), we introduce a new file structure called a Relational Inverted Structure (RIS). The design of RIS is aimed at rapid processing of relational operations and a unified maintenance facility. In RIS file organization, the concept of 'pseudo' relational operations is introduced. The pseudo relational operation is performed on not actual data values but on hashed values of the original data. The hashed attribute values are used to eliminate unnecessary data during the search of RIS files, which are proved not to be in the answer. These pseudo operations are useful to decrease the number of accessions to relations, and to achieve rapid retrieval.

The problem of decreasing required storage space is discussed in this thesis from views of both data dependency theory and physical data organization. When decreasing the required storage space, it is important to consider the physical data organization which does not lose the retrieval time performance and the ease of update maintenance. The following problem are discussed in this thesis:

(c) How to decompose relations: a relation is sometimes possible to be decomposed into smaller relations whose total required storage space is smaller than that of the original relation. Codd's functional dependency provides a sufficient condition for a relation to be decomposed into two smaller relations without loss of information. In this sense, data dependency theory is useful to reduce the required storaghe space. Multivalued dependencies [FAGI7709] [ZANI7607] and join dependencies [RISS7809] are also integrity constraints which are generalized from functional dependencies. These generalized dependencies provides necessary and sufficient conditions of information lossless decomposition of relations. Several problems of these generalized dependencies are remained unsolved.

(d) How to compress data: This problem is concerned with the problem in (a). It is important to compress relations and other additional data files, such as secondary indexes, in a way that their retrieval time performance is not decreased.

As for (c), we mainly investigate the properties of multivalued dependencies. Several results will be summarized in the problem of update maintenance described later. As for (d), our quasi-consecutive retrieval file organization is useful to store relations and secondary indexes with less redundancy. The file organization does not decrease the retrieval time

performance so much.

The problem of update maintenance is mainly concerned with how to design relations suitable for maintaining given integrity constraints. Data dependency theory is useful to handle dependency constraints. In this thesis, the following problems are discussed:

(e) How to specify and maintain multivalued dependencies: Multivalued dependencies are useful integrity constraints to handle many-to-many relationships in the relational data model. However, their specification is not so easy since the validity of multivalued dependencies depends on the underlying set of attributes of the relation. That is, some multivalued dependency holds in a certain set of attributes, but may not hold in its superset of attributes. The multivalued dependency, which holds in a subset of attributes of a relation, is called an embedded multivalued dependency. The properties of embedded multivalued dependencies are not well known. Furthermore, if we can reduce some set of multivalued dependencies into a set of embedded multivalued dependencies concerned with a smaller number of attributes, it is possible to reduce the cost of maintaining these dependency constraints.

(f) How to obtain update independence of relations: When we have several relations, it is desirable that each relation can be updated independently from others. That is, each relation is updated according to only the integrity constraints of the relation. However, there has not been any tool to analyze what data dependencies can be enforced totally by a set of relations when we allow the update independence of relations. Most of conventional results in data dependency theory implicitly or explicitly assumed that every relation must be a part of a common universal relation at any time.

As for (e), in this thesis, we clarify several new properties of embedded multivalued dependencies. For example, a necessary and sufficient condition for a embedded multivalued dependency, which holds in a certain set of attributes, to hold in its superset of attributes is provided. Furthermore, in order to make the specification of multivalued dependencies easy, we investigate the semantic aspects of data dependencies. Several unknown properties of multivalued dependencies are clarified on their semantic aspects.

As for (f), we provide a necessary and sufficient condition for a set of relations to enforce a data dependency under a certain assumption on relations. By this result, we can obtain the update independence of relations in a way that a set of relations totally preserves a given set of integrity constraints.

1.3 Outline of the Thesis

In Chapter 2, we summarize basic concepts of the relational data model, such as relations, relational algebra operators, integrity constraints, schemes, data dependencies and normal forms.

Chapter 3 is concerned with the properties of embedded multivalued dependencies [TANAK7908] [KAMBT7911R] [KAMBT7801] [TANAK7901] [TANAK7808]. Several new properties of embedded multivalued dependencies are provided. Especially, a basic theorem about the interaction between multivalued and embedded multivalued dependencies is shown. This leads to a necessary and sufficient condition for an embedded multivalued dependency to hold in a larger set of attributes. Some useful

equivalence relationships between two sets of multivalued and embedded multivalued dependencies are also shown.

Chapter 4 is concerned with the problem what data dependencies are enforced by a set of relations [TANAK7904] [KAMBT7911R] [TANAK8002] [TANAK7907] [TANAK8005]. The notion of 'preservation' of data dependencies by a relational database scheme is introduced. We allow each relation to be updated independently from others. Under a certain assumption on relations, a necessary and sufficient condition for a functional, an embedded multivalued or an embedded join dependency to be preserved by a relational databse scheme is provided.

Chapter 5 is concerned with semantic aspects of data dependencies [KAMBT7710] [KAMBT7911S] [KAMBT7709] [KAMBT7805] [TANAK7710] [KAMBT7710P] [KAMBT7710H] [TANAK7803] [KAMBT7808] [KAMBT7810]. Especially, the semantic aspects of functional and multivalued dependencies are discussed. We clarify, for example, that a multivalued dependency does not always capture many-to-many relationships between attributes. Moreover, some transitively specified multivalued dependencies are shown to often impose a semantically unnatural constraint. A sufficient condition for these multivalued dependencies to hold in a 'natural' sense is also provided.

Chapter 6 is concerned with the organization of Quasi-Consecutive Retrieval files [TANAL79] [TANAK7703] [TANAK7510] [KAMBT7603] [TANAK7611]. Mainly, graph theoretical properties of Quasi-Consecutive Retrieval files are discussed. We porvide a necessary and sufficient condition for the existence of a Quasi-Consecutive Retrieval file without redundancy. Furthermore, a heuristic computer algorithm is given, which organizes a Quasi-Consecutive Retrieval file with

less redundancy.

Chapter 7 is concerned with the organization of Relational Inverted Structure files [TANAL7808] [LEVIK7904] [LEVIK7911] [YAJIK8001] [LEVIK8002] [TANAY8003] [LEVII8003] [TANAL7710] [TANAL7808I] [TANAL7810] [TANAY7810] [LEVIT7810]. The organization of Relational Inverted Structure files and query processing algorithms by using this file are provided.

Chapter 8 is a conclusion of this thesis. Main contributions and future problems are summarized.

CHAPTER 2 RELATIONAL DATA MODEL

In this chapter, we summarize basic concepts of the relational data model introduced by E.F. Codd [CODD7006]: relations, relational algebra operators, integrity constraints, schemes, data dependencies and normal forms.

2.1 Relations

The mathematical concept underlying the relational data model is the set-theoretic relation. Two views of relations are often used [ULLM80]: a set-of-tuples view and a set-of-mappings view. In this thesis, according to [ULLM80], formally we use the set-of-mapping view for relations, but for convenience, we display relations as sets of tuples. In both cases, mathematically, a relation may be an infinite set, but as far as databases are concerned, we assume that every relation is finite through out this thesis.

Codd used a set-of-tuples view in his original literature [CODD7006] as follows:

Definition 2.1: [CODD7006]
A relation is a finite subset of the Cartesian product of a list of (not necessarily distinct) domains. A domain is simply a set of all the possible values, which can appear as the corresponding attribute values. The Cartesian product of domains $D_1,\ldots,D_n$, denoted by

$$D_1 \times D_2 \times \ldots D_n,$$

is the set of all n-tuples $(v_1,v_2,\ldots,v_n)$ such that for all i,

$1 \le i \le n$, v_i is in domain D_i. The members of a relation are called tuples. The domains are often given names, called attributes which are distinct within a relation.

Conceptually, a relation can be viewed as a table where each row is a tuple and each colume corresponds to an attribute. The ordering of rows is immaterial, and all rows of a table are distinct.

Aho et al. introduced a set-of-mapping view for a relation [AHO-B7909] as follows:

Definition 2.2: [AHO-B7909]
A relation is defined as a finite set of mappings from the set of attributes to the corresponding domains. Attributes are symbols taken from a given finite set of symbols. We use A,B,C,... (possibly with subscripts) to denote single attributes, and ...,X,Y,Z (possibly with subscripts) to denote sets of attributes. For a set of attributes $R=\{A_1,A_2,\ldots,A_n\}$ and an associated domain D_i for each attribute A_i, a relation r on R is a finite set of mappings t such that

$$t: \{A_1,A_2,\ldots,A_n\} \longrightarrow D,$$

where D is the union of the D_i's. The mapping must map each attribute in R to a member of its corresponding domain. That is,

$$t(A_i) \in D_i$$

should hold for all i. $1 \le i \le m$. The value $t(A_i)$ is called a A_i-value of t. If we assume some ordering in a set of

attributes $X=\{A_{i+1},\ldots,A_{i+m}\}$, then the value $(t(A_{i+1}),\ldots,t(A_{i+m}))$ is called a X-value of t.

Example 2.1: Let R={ID, AUTHOR, KEYWORD, PUBLICATION, DATE} be a given set of attributes for a bibliographic database. For each attribute A, we denote its corrsponding domain by DOM(A). The meanings of these attributes and their corresponding domains are, for example, as follows:

Attribute	Meaning	Domain
ID	ID codes	DOM(ID)={1,2,3...}
AUTHOR	Author's name	DOM(AUTHOR)= {Aho,Beeri, Codd,...}
KEYWORD	Keyword	DOM(KEYWORD)= {FD, MVD, join,...}
PUBLICATION	Publication name	DOM(PUBLICATION)= {ACMTODS,CACM,...}
DATE	Published date	DOM(DATE)= {1970/06,1977/09,...}

An example relation r on R is shown in a tabular form in Fig.2.1. For example, the first row (or called tuple) corresponds to the mapping t_1 in r such that

t_1(ID)=1,
t_1(AUTHOR)='Aho',
t_1(KEYWORD)='join',
t_1(PUBLICATION)='ACMTODS',
t_1(DATE)='1979/09'.

ID	AUTHOR	KEYWORD	PUBLICATION	DATE
1	Aho	join	ACMTODS	1979/09
1	Beeri	join	ACMTODS	1979/09
2	Codd	relation	CACM	1970/06
1	Ullman	join	ACMTODS	1979/09
3	Fagin	fd	ACMTODS	1977/09
3	Fagin	mvd	ACMTODS	1977/09
2	Codd	fd	CACM	1970/06

Fig.2.1. An example relation r on R.

2.2 Relational Algebra Operators

In this section, we define basic operators of the relational algebra [CODD7105R], [AHO-B7909], [BEERV7911], [ULLM80].

Definition 2.3: For a mapping t on a set R of of attributes and a set X R, we denote the restriction of t to X by t[X]. Let r be a relation on R. The projection of r on X, denoted by r[X], is the set

$$r[X]=\{t' \mid \text{for some } t \in r,\ t'=t[X]\}.$$

The projection r[X] is a relation on X whose elements are the restrictions of the mappings in r to X. Furthermore, we denote a projection of a subrelation of r, in which every tuple's X-value

is equal to x, as follows:

$$r[x,Y]=\{t' \mid \text{for some } t \in r,\ t'=t[Y] \text{ and } t(X)=x\}$$

Intuitively, the projection r[X] is obtained from the table representing r by deleting all columns labeled with attributes in R-X, and identifying common rows.

The (natural) join operation is an operation that combines several relations into one relation as follows:

Definition 2.4: For $1\le i\le n$, let r_i be a relation on the set R_i of attributes. The (natural) join of $r_1, r_2, \ldots, r_n$, denoted by

$$r_1 * r_2 * \ldots * r_n \text{ or } \overset{n}{\underset{i=1}{*}} r_i,$$

is the relation on $\overset{n}{\underset{i=1}{U}} R_i$ defined by

$$\overset{n}{\underset{i=1}{*}} r_i = \{t \mid t \text{ is a mapping on } \overset{n}{\underset{i=1}{U}} R_i \text{ such that } t[R_i] \text{ is in } r_i \text{ for all } i,\ 1\le i\le n\}.$$

Each tuple t in the join $\overset{n}{\underset{i=1}{*}} r_i$ is generated from tuples $t_i \in r_i$ such that every two of them map the attributes common to them to the same values. Generally, if we project $\overset{n}{\underset{i=1}{*}} r_i$ on some R_j, that is, if we take $(\overset{n}{\underset{i=1}{*}} r_i)[R_j]$, then we obtain a subset of r_j. The (natural) join operator satisfies the following properties:

(1) The join is commutative. That is, for any $1\le i,j\le n$,

$$r_i * r_j = r_j * r_i.$$

(2) The join is associative. That is, for any $1 \leq i,j,k \leq n$

$$(r_i * r_j) * r_k = r_i * (r_j * r_k).$$

(3) If for some i and j, $R_i \cap R_j \neq \emptyset$ but $r_i[R_i \cap R_j] \cap r_j[R_i \cap R_j] = \emptyset$, then the join is empty.

Example 2.2: Consider a bibliographic relation r shown in Fig.2.1. Let X={ID, AUTHOR} and Y={ID, PUBLICATION} be subsets of R. The projections r[X] and r[Y] are shown in tabular forms in Fig.2.2(a) and (b), respectively.

(a) r[X]:

ID	AUTHOR
1	Aho
1	Beeri
2	Codd
1	Ullman
3	Fagin

(b) r[Y]:

ID	PUBLICATION
1	ACMTODS
2	CACM
3	ACMTODS

Fig.2.2. Example projections on X={ID,AUTHOR} and on Y={ID,PUBLICATION}

The join of the relations r[X] and r[Y] shown above is also shown in Fig.2.3.

ID	AUTHOR	PUBLICATION
1	Aho	ACMTODS
1	Beeri	ACMTODS
2	Codd	CACM
1	Ullman	ACMTODS
3	Fagin	ACMTODS

Fig.2.3. A join of r[X] and r[Y].

2.3 Integrity Constraints and Schemes

Definition 2.5: Let U be a given finite set of attributes. A relation scheme R over U is a subset of the set U of attributes. A (relational) database scheme on U is a set of relation schemes $\{R_1, R_2, \ldots, R_p\}$, where $\bigcup_{i=1}^{p} R_i = R$. A (relational) database is a set of relations $r_1, r_2, \ldots, r_p$, where r_i is a relation on relation scheme R_i.

Hereafter, we often represent a set of attributes by omitting commas and set braces, so that, for example, $\{A_1, \ldots, A_n\}$ is written $A_1A_2\ldots A_n$. The union of sets of attributes is also represented by concatenation, e.g., X U Y U Z is written XYZ.

The contents of a relational database is time-varying since several update operations are allowed to be performed, such as insertions, deletions and updates of tuples. For any given application, however, only a subset of the set of all the possible databases is usually of interest. This subset is

defined by 'integrity constraints', which are specified for the application.

Definition 2.6: An integrity constraint c for a relation scheme R is a predicate, that assigns to each relation on R either the value T (True) or the value F (False). When a relation r is assigned to the value T, r is said to satisfy c, and c is said to hold in r. For a given set C of integrity constraints of a relation scheme R, we denote the set of relations on R that satisfy every constraint in C by SAT(C). For a singleton set {c} , we denote it by SAT(c) instead of SAT({c}). A set C of integrity constraints is said to hold in a relation scheme R if SAT(C) are the only set of relations on R that are of interest.

In other terms, integrity constraints are used to specify a set of allowable update operations for each database. When a relation r satisfies some integrity constraints, and some update operation is to change r into r', the update operation is allowed if and only if r' satisfies the integrity constraints.

2.4 Data Dependencies

Several integrity constraints concerned with attribute relationships have been introduced as data dependencies, such as functional dependencies and multivalued dependencies.

Codd introduced the concept of functional dependencies as defined below:

Definition 2.7: A functional dependency (for short, FD) is a statement X -> Y. It is defined for every relation on R such

that $XY \subseteq R$. The FD X -> Y holds in the relation r if and only if, for all mappings t_1, t_2 in r, $t_1[X]=t_2[X]$ implies $t_1[Y]=t_2[Y]$. That is, the FD X -> Y holds in r if and only if every two tuples of r that have the same X-value also have the same Y-value. If $X \supseteq Y$, then the FD X -> Y is called a trivial FD; otherwise a nontrivial FD. When we wish to declare explicitly that X -> Y does not hold, we denote it by X $\not\rightarrow$ Y.

Schmid and Swenson [SCHMS7505] showed several problems of functional dependencies. In order to solve some of these problems, Fagin [FAGI7709] and independently, Zaniolo [ZANI7607] introduced a new dependency called a multivalued dependency as follows:

Definition 2.8: A multivalued dependency (for short, MVD) is a statement X ->-> Y, that is a generalization of the notion of FDs. It is defined for every relation on R such that $XY \subseteq R$. Let Z=R-XY. The MVD X ->-> Y holds in the relation r if and only if, for all mappings t_1, t_2 in r such that $t_1[X]=t_2[X]$, there exists a mapping t in r such that

$$t[X]=t_1[X]=t_2[X],$$
$$t[Y]=t_1[Y],$$
$$t[Z]=t_2[Z].$$

If Z is an empty set or $X \supseteq Y$, then X ->-> Y is called a trivial MVD; otherwise, a nontrivial MVD.

Note that any FD X -> Y on R is also an MVD X ->-> Y from the definition of MVDs. In this sense, The notion of MVDs is a generalization of FDs. The MVDs are defined for the set of all

the attributes of a relation scheme. Fagin also introduced the notion of embedded multivalued dependencies, which is a generalization of MVDs, as follows:

Definition 2.9: An embedded multivalued dependency (for short, EMVD) is a statement X ->-> Y|Z. It is defined for every relation on R such that XYZ $\subseteq$ R and Y $\cap$ Z $\subseteq$ X. Let r[XYZ] be a projection of a relation r of a relation scheme R. The EMVD X ->-> Y|Z holds in the relation r if and only if the MVDs X ->-> Y and X ->-> Z hold in r[XYZ]. If X $\supseteq$ Y or X $\supseteq$ Z, then the EMVD X ->-> Y|Z is called a trivial EMVD; otherwise a nontrivial EMVD. If XYZ=R and Z=R-XY, then the EMVD X ->-> Y|Z is equivalent to the MVD X ->-> Y and X ->-> Z on R.

The EMVD X ->-> Y|Z provide a necessary and sufficient condition for a projection r[XYZ] to be decomposable losslessly into r[XY] and r[XZ]. That is, the EMVD X ->-> Y|Z holds in r if and only if

r[XYZ]=r[XY]*r[XZ]

(r[XYZ] is obtained as a natural join of r[XY] and r[XZ]). Rissanen introduced the notion of join dependencies, which is a generalization of MVDs [RISS7809]. The join dependency provides a necessary and sufficient condition for a relation to be decomposable losslessly into n projections of r. The definition of join dependencies is as follows:

Definition 2.10: A join dependency (for short, JD) is a statement *[R_1,R_2,...,R_n]. It is defined for every relation on

R such that $R=\bigcup_{i=1}^{n} R_i$. The JD $*[R_1,R_2,\ldots,R_n]$ holds in the relation r if and only if

$$\overset{n}{*}_{i=1} r[R_i]=r$$

holds.

The embedded version of the JDs is defined as follows:

Definition 2.11: An embedded join dependency (for short, EJD) is a statement $*[R_1,R_2,\ldots,R_m]$, which is defined for every relation on R such that $R \supseteq \bigcup_{i=1}^{m} R_i$. The EJD $*[R_1,R_2,\ldots,R_m]$ holds in the relation r if and only if

$$\overset{m}{*}_{i=1} r[R_i]=r[\overset{m}{\bigcup}_{i=1} R_i].$$

Fig.2.4 shows an example relation of a relation scheme R={EMPLOYEE, SALARY, CHILD, PROJECT, MANAGER}. Here, we assume that each employee has exactly one salary and may have one or more children. Each employee may belong to one or more projects and each project is managed by one or more managers. From these assumptions, we see that this relation scheme R obeys the following data dependencies:

EMPLOYEE -> SALARY (FD)
EMPLOYEE ->-> CHILD (MVD)
PROJECT ->-> MANAGER (MVD)
EMPLOYEE ->-> PROJECT|{SALARY,CHILD} (EMVD)
$*[R_1,R_2,R_3,R_4]$ (JD)
where R_1={EMPLOYEE,SALARY}
R_2={EMPLOYEE,CHILD}

$$R_3 = \{EMPLOYEE, PROJECT\}$$
$$R_4 = \{PROJECT, MANAGER\}$$

Note that the MVD EMPLOYEE ->-> PROJECT does not hold in this relation r, while the EMVD EMPLOYEE ->-> PROJECT|{SALARY,CHILD} holds in r.

EMPLOYEE	SALARY	CHILD	PROJECT	MANAGER
e_1	s_1	c_1	p_1	m_1
e_1	s_1	c_2	p_1	m_1
e_1	s_1	c_1	p_1	m_2
e_1	s_1	c_2	p_1	m_2
e_2	s_2	c_3	p_1	m_1
e_2	s_2	c_3	p_2	m_2
e_2	s_2	c_3	p_1	m_2
e_2	s_2	c_3	p_2	m_3

Fig.2.4. An example relation r on R.

It is important to distinguish a dependency satisfied by some relations from the one satisfied by some relation schemes. If we say a dependency d holds in a relation scheme R, then it means that the set of all the allowable relations on R is equal to SAT(d). When a set of dependencies $D=\{d_1,\ldots,d_n\}$ is said to hold in R, the set of all the allowable relations on R is

$$SAT(D)=SAT(d_1) \cap \ldots \cap SAT(d_n).$$

Some dependencies, which do not belong to D, may also hold in any relation in SAT(D). The relationship between these

dependencies and D is captured by the 'implication' relationship defines as follows:

Definition 2.12: Let d be a single dependency that is defined on R. The dependency d is said to be implied by a set D of dependencies on R if and only if d holds in any relation on R that satisfies all the dependencies in D. If D implies d, then we denote it by D $\models$ d.

2.5 Normal forms

One important problem of the relational database design is how to obtain relation schemes in which so called storage anomalies [DATE77] are avoided as much as possible. Storage anomalies are the undesirable situations, when updating operations are performed, which causes the inconsistency of data. In order to avoid storage anomalies as much as possible, Codd introduced a series of normal forms, such as first, second, third and Boyce-Codd normal forms based on the notion of FDs. Furthermore Fagin also introduced the notion of fourth normal form for relation schemes based on MVDs.

Definition 2.13: A relation is in first normal form (for short, 1NF) if ant only if each attribute value is atomic, that is each attribute value is neither a set nor a relation. Given a relation scheme R, we say that a subset X of R is a key if and only if the FD X -> R is implied by the dependencies on R and there is no proper subset X' of X such that X' -> R is implied. A relation scheme R in 1NF is in Boyce-Codd normal form (for

short, BCNF) if and only if for each FD d of R, the FD X -> R holds in R such that X -> R implies d and X is a key of R. A relation scheme R in 1NF is in fourth normal form (for short, 4NF) if and only if ,for each MVD that holds in R, the FD X -> R holds in R such that X -> R implies d and X is a key of R.

For example, several storage anomalies may occur in the relation r shown in Fig.2.4. If we want to change the SALARY-value s_1 of the tuple (e_1,s_1,c_1,p_1,m_1) into s_3, then we should also change the SALARY-value of the three tuples (e_1,s_1,c_2,p_1,m_1), (e_1,s_1,c_1,p_1,m_2) and (e_1,s_1,c_2,p_1,m_2); othwise, the contents of the relation r becomes inconsistent since the FD EMPLOYEE -> SALARY would not hold in r. If we want to insert a tuple (e_1,s_1,c_4, p_1,m_1) into r in order to add the employee e_1's child c_4, we should also insert the tuple (e_1,s_1,c_4,p_1,m_2) since each EMPLOYEE-value in r should be associated with a set of CHILD-values in a way that does not depend on other attribute values. If we want to delete the tuple (e_2,s_2,c_3,p_2,m_2) from r in order to change the employee e_2's projects from p_1,p_2 into p_1 , then we should also delete the tuple (e_2,s_2,c_3,p_2,m_3). In this case, however, the information that the project p_2 is managed by m_2 and m_3 is also lost. Although this deletion does not bring the inconsistency of data, it is also included in storage anomalies since other basic information unit is lost by the deletion of the tuple.

As shown in Fig.2.5, if we represent R={EMPLOYEE, SALARY, CHILD, PROJECT, MANAGER} by the following four relation schemes:

R_1={EMPLOYEE,SALARY},
R_2={EMPLOYEE,CHILD},
R_3={EMPLOYEE,PROJECT},

$R_4 = \{PROJECT, MANAGER\}$,

then the storage anomalies above will not occur. Furthermore, the original relation r can be obtained by taking a natural join of r_1, r_2, r_3 and r_4. The key of R_1 is EMPLOYEE and the key of R_3 is {EMPLOYEE, PROJECT}. Since every dependency in each relation scheme is a result of the key of the scheme, all the relation schemes are in BCNF and 4NF.

EMPLOYEE	SALARY
e_1	s_1
e_2	s_2

r_1

EMPLOYEE	CHILD
e_1	c_1
e_1	c_2
e_2	c_3

r_2

EMPLOYEE	PROJECT
e_1	p_1
e_2	p_1
e_2	p_2

r_3

PROJECT	MANAGER
p_1	m_1
p_1	m_2
p_2	m_2
p_2	m_3

r_4

Fig.2.5. Example relations on R_1, R_2, R_3 and R_4.

CHAPTER 3 PROPERTIES OF EMBEDDED MULTIVALUED DEPENDENCIES

Fagin and independently, Zaniolo introduced the notion of a multivalued dependency (MVD). The definition of MVDs refers to an underlying set of attributes of a relation. The embedded multivalued dependency (EMVD), which is also introduced by Fagin, is intuitively an MVD that holds in a projection of an original relation on the subset of attributes of the relation. The properties of EMVDs are not well known although EMVDs play an important role in designing a relational database scheme by MVDs.

The main results of this chapter are the following [TANAK7908]: (1) A basic theorem about the interaction between MVDs and EMVDs is provided. Several useful inference rules for MVDs and EMVDs are derived from this theorem. (2) A marked Hasse diagram called a dependency diagram is introduced to investigate the interactions between MVDs and EMVDs. (3) We provide some conditions for an MVD or a set of MVDs to hold after the addition or deletion of some attributes. (4) Some useful equivalence relationships between two sets of MVDs and EMVDs are provided. We also show some conditions to represent a given set of MVDs and EMVDs in a reduced form.

3.1 Introduction

One essential problem of the relational database design is how to obtain a set of relation schemes which are suitable for maintaining given dependency constraints. The properties of functional and multivalued dependencies and the properties of the interactions between them have been already well studied [CODD7105F], [ARMS7408], [FAGI7709], [ZANI7607], [BEERF7708].

The properties of embedded multivalued dependencies, however, have not been sufficiently studied although embedded multivalued dependencies play an important role in designing a relational database scheme [KAMBT7801] [KAMBT7911R].

In this chapter, we mainly discuss interactions between embedded multivalued dependencies and multivalued dependencies. Several properties of embedded multivalued dependencies are provided.

The concept of MVDs leads to the normalization of a relation scheme, which decompose the relation scheme into a set of fourth normal form (for short, 4NF) relation schemes [FAGI7709]. The normalization is useful to avoid so called storage anomalies [DATE77] as shown in Section 2.5. Fagin and independently, Zaniolo also introduced a decomposition approach for designing 4NF relation schemes from an initial relation scheme. The input dependencies are restricted only to FDs and MVDs that hold in the initial relation scheme. The problems, which we are faced with in using their decomposition approach, are the following:

(1) How to specify 'correct' multivalued dependencies for the initial relation scheme (Context Dependence Problem of MVDs)?

(2) How to examine whether or not the given dependencies are completely reflected on the obtained 4NF relation schemes (Equivalence Testing Problem)?

(3) How to store the dependency constraints which are not reflected on the obtained relation schemes if they exist (Minimal Representation Problem of MVDs)?

As for (1), we should note that it depends on the underlying set of attributes whether an MVD holds in a relation scheme. It is not so easy to specify correct MVDs when a relation scheme consists of a large number of attributes. In

this chapter, we show a necessary and sufficient condition for an MVD, which holds in R', to hold in R, where R' is an arbitrary proper subset of R.

As for (2), it is necessary to examine whether or not the following two sets of dependencies are equivalent: One is a given set of FDs and MVDs for the initial relation scheme. The other is a set of FDs and MVDs which are enforced by the obtained 4NF relation schemes. Our results in this chapter are useful to analyse this representation problem. Furthermore, detailed discussions will be provided for this problem in Chapter 4.

As for (3), it is useful to represent a set of FDs and MVDs, that are not reflected, in a reduced form. In this chapter, we discuss the redundancy of a set of MVDs and EMVDs, and the reducibility of a dependency (MVD or EMVD) within a given set of dependencies. Some conditions to reduce a dependency into another dependency having a smaller number of attributes are provided.

In order to handle these problems, we also introduce the dependency diagram, which is a marked Hasse diagram. This diagram is useful to investigate the interaction between MVDs and EMVDs.

3.2 Basic Properties of Embedded Multivalued Dependencies

Fagin and independently, Zaniolo studied the projectability of MVDs in [FAGI7709] and [ZANI7607], respectively. For example, assume that the MVD A ->-> BC holds in R=ABCDE. Let R_1=ABD be a relation scheme for which any relation on R_1 is a projection of a relation on R. The example relations r on R and

r_1 on R_1 are given in Fig.3.1(a) and (b), respectively. It is easy to verify that the MVD A ->-> B holds in r_1. That is, the MVD A ->-> BC of R=ABCDE projects down into the MVD A ->-> B of R_1=ABD. The MVD A ->-> B, however, does not hold in r, conversely. In order to handle this context dependence problem of MVDs, Fagin introduced the notion of the embedded multivalued dependency (EMVD). In this example, A ->-> B|D is an EMVD that hold in both r and r_1.

Fagin and Zaniolo obtained a useful inference rule to derive EMVDs that are implied by a given MVD. Furthermore, Aho et al. showed how to compute the projection of a set of FDs and MVDs as follows:

(a)

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_2	c_2	d_1	e_1
a_1	b_1	c_1	d_2	e_1
a_1	b_2	c_2	d_2	e_1
a_2	b_1	c_1	d_1	e_2
a_2	b_2	c_2	d_1	e_2

r

A	B	D
a_1	b_1	d_1
a_1	b_2	d_1
a_1	b_1	d_2
a_1	b_2	d_2
a_2	b_1	d_1
a_2	b_2	d_1

$r_1 = r[R_1]$

Fig.3.1. Example relation and its projection.

Theorem 3.1: [AHO-B7909]
Let D be a set of FDs and MVDs that hold in a relation scheme R. For any subset S of R, the set of FDs and MVDs, which hold in S, is computed in the following way:

(1) Compute the closure D^+ of D, that is a set of all the FDs and MVDs implied by D.

(2) For each X -> Y in D^+, if $X \subseteq S$, then X -> Y $\cap$ S holds in S.

(3) For each X ->-> Y in D^+, if $X \subseteq S$, then X ->-> Y $\cap$ S holds in S.

(4) No other FDs or MVDs are implied by the fact that D holds in R.

The converse of (3) in Theorem 3.1 does not hold while the converse of (2) in Theorem 3.1 holds. That is, even if X ->-> Y $\cap$ S holds in S, the MVD X ->-> Y does not hold in R. The example shown above is a counterexample for the converse of (3) in Theorem 3.1.

Several complete sets of inference rules for FDs and MVDs are known, which are useful to compute the closure D^+ of D in Theorem 3.1. Beeri et al. [BEERF7708] provided a complete set of inference rules for MVDs. Every MVD implied by a given set of MVDs is derived by some combination of the following four inference rules:

(MVD0) Complementation rule:
X ->-> Y holds in R if and only if X ->-> Z holds in R, where Y $\cap$ Z $\subseteq$ X and R=XYZ.
(MVD1) Reflexivity rule:
If Y $\subseteq$ X, then X ->-> Y holds in R, where R is an arbitrary relation scheme such that R $\supseteq$ XY.
(MVD2) Augmentation rule:
If Z $\subseteq$ W and X ->-> Y holds in R, then XW ->-> YZ also holds in R, where R $\supseteq$ XYZW.
(MVD3) Transitivity rule:

If X ->-> Y and Y ->-> Z hold in R, then X ->-> Z-Y also holds in R, where R $\supseteq$ XYZ.

The inference rules listed above can be easily generalized to the inference rules for EMVDs as follows:

(EMVD0) Complementation rule:
If X ->-> Y|Z holds in R, then X ->-> Z|Y holds in R, where R $\supseteq$ XYZ.
(EMVD1) Reflexivity rule:
If Y $\subseteq$ X, then X ->-> Y|Z holds in R, where R $\supseteq$ XYZ.
(EMVD2) Augmentation rule:
If X ->-> Y|ZW holds in R, then XW ->-> Y|ZW' holds in R, where R $\supseteq$ XYZW and W $\supseteq$ W'.
(EMVD3) Transitivity rule:
If X ->-> Y|ZW and Y ->-> Z|XW hold in R, then X ->-> Z-Y|YW holds in R, where R $\supseteq$ XYZW.

The result in Theorem 3.1 can be restated as the following inference rule for EMVDs [FAGI7709] [ZANI7607]:

(EMVD4) Projection rule:
If X ->-> Y|Z holds in R, then X ->-> Y'|Z' holds in R, where R $\supseteq$ XYZ, X $\supseteq$ X' and Y $\supseteq$ Y'.

We now show a basic theorem on interactions between MVDs and EMVDs.

Theorem 3.2: Both the MVD XY ->-> Z and the EMVD X ->-> Y|Z hold in R if and only if the MVD X ->-> Z holds in R, where R is an arbitrary relation scheme such that R $\supseteq$ XYZ and Y $\cap$ Z $\subseteq$ X.

(Proof). Assume that W=R-XYZ and $W \neq \phi$. By the assumption of the existence of the EMVD X ->-> Y|Z, for each XY-value xy in r[XY], r[x,Z]=r[xy,Z] holds. Furthermore, by the assumption of the MVD XY ->-> Z, for each XYW-value xyw in r[XYW], r[xy,Z]=r[xyw,Z] holds. Consequently, for each XYW-value xyw in r[XYW], r[x,Z]=r[xyw,Z] holds. Therefore, the MVD X ->-> Z holds in R. Conversely, assume that the MVD X ->-> Z holds in R. By Theorem 3.1, the EMVD X ->-> Y|Z is implied by X ->-> Z. Furthermore, by the augmentation rule (MVD2), the MVD XY ->-> Z is also implied by X ->-> Z. Q.E.D.

The result in Theorem 3.2 can be further generalized, and we can obtain the following inference rule for EMVDs:

(EMVD5) Joinability rule:
X ->-> Z|YW holds in R if and only if XY ->-> Z|W and X ->-> Y|Z hold in R, where $R \supseteq XYZW$.

Here, the name of our EMVD5 rule is given by Parker et al. [PARKP8005].

Both Theorem 3.1 and Theorem 3.2 are also useful to consider conditions such that a given MVD of a relation scheme holds after some addition or deletion of attributes are performed. The following corollary provides a necessary and sufficient condition such that an MVD, which holds in a relation scheme, also holds after some deletion of attributes.

Corollary 3.1: Let R, R', X and Y be sets of attributes such that $R \supseteq R' \supseteq XY$ and $X \cap Y = \phi$. Let X ->-> Y be an arbitrary MVD which holds in R'. Then, X ->-> Y holds in R if and only if the MVD R'-Y ->-> Y holds in R.

(Proof). As for the sufficiency, assume that the MVD R'-Y ->-> Y holds in R. It can be restated into that the EMVD

$$X \cup (R'-X-Y) \text{ ->-> } Y|R-R'$$

holds in R. The MVD X ->-> Y on R' implies the EMVD

$$X \text{ ->-> } R'-X-Y|Y.$$

By using Theorem 3.2 and the above two EMVDs, we obtain the EMVD

$$x \text{ ->-> } Y|(R-R') \cup (R'-X-Y).$$

The above EMVD is equal to the EMVD X ->-> Y|R-X-Y. Therefore, the MVD X ->-> Y holds in R. As for the necessity, assume that the MVD X ->-> Y holds in R. By the augmentation rule (MVD2), we obtain the MVD

$$X \cup (R'-X-Y) \text{ ->-> } Y$$

that is eauql to the MVD

$$R'-Y \text{ ->-> } Y.$$

Q.E.D.

In Corollary 3.1, it is not necessarily guaranteed that another complementary MVD X ->-> R'-X-Y also holds in R. The following corollary provides a necessary and sufficient condition such that both X ->-> Y and X ->-> R'-X-Y, which hold in R', also hold in R.

Corollary 3.2: Let X, Y, Z and W be disjoint sets of attributes such that R=XYZW. Let the MVDs X ->-> Y and X ->-> Z hold in a relation scheme R'=XYZ. Then, the MVDs X ->-> Y and X ->-> Z hold in R=XYZW if and only if the MVD X ->-> YZ holds in R.

(Proof). From the assumption that X ->-> YZ holds in R, we obtain XY ->-> Z and XZ ->-> Y by using the augmentation rule (MVD2). Since X ->-> Y|Z holds in R', by Theorem 3.2, we obtain X ->-> Z and X ->-> Y, which hold in R, from XY ->-> Z and XZ ->-> Y, respectively.

In [BEERF7708], Beeri et al. showed that if both X ->-> Y_1 and X ->-> Y_2 hold in R, then X ->-> Y_1Y_2 also holds in R. Applying this rule to X ->-> Y and X ->-> Z, we obtain X ->-> YZ. Q.E.D.

Example 3.1: Let R'=ABC be a relation scheme such that any relation on R' is a projection of some relation on R=ABCD. Assume that the EMVD A ->-> B|C holds in R and in R'. By Corollary 3.1 and Corollary 3.2, we obtain the following:

(1) The MVD A ->-> B holds in R if and only if AC ->-> B holds in R.

(2) The MVD A ->-> C holds in R if and only if AB ->-> C holds in R.

(3) Both the MVDs A ->-> B and A ->-> C hold in R if and only if A ->-> BC holds in R.

The following corollary is a generalization of the EMVD5 rule. Its proof is straightforward from the EMVD5 rule and the EMVD2 rule. But, the corollary will be a useful inference rule to derive new EMVDs from a set of EMVDs:

Corollary 3.3: Assume that both X_1 ->-> $Y_1|Z_1$ and X_2 ->-> $Y_2|Z_2$ hold in R, where $R \supseteq X_1Y_1Z_1$, $X_1Y_1=X_2Y_2Z_2$ and X_1,Y_1,Z_1 are mutually disjoint, and X_2,Y_2,Z_2 are also disjoint. Then,

$$X_2(Y_2-Y_1) \text{ ->-> } Y_1 \cap Y_2|Z_1Z_2$$

holds in R.

(Proof). X_1 ->-> $Y_1|Z_1$ is restated as

$$X_1 \text{ ->-> } (Y_1 \cap Y_2)(Y_1-Y_2)|Z_1.$$

By the MVD2 rule, we obtain

$$X_1(Y_1-Y_2) \text{ ->-> } Y_1 \cap Y_2|Z_1.$$

Since $X_1Y_1=X_2Y_2Z_2$ and $X_1(Y_1-Y_2)-(Y_1 \cap Y_2)= X_2(Y_2-Y_1)Z_2-(Y_1 \cap Y_2)$, the above EMVD is also restated as

$$X_2(Y_2-Y_1)Z_2 \text{ ->-> } Y_1 \cap Y_2|Z_1.$$

On the other hand, X_2 ->-> $Y_2|Z_2$ is equal to

$$X_2 \text{ ->-> } (Y_2-Y_1)(Y_2 \cap Y_1)|Z_2.$$

By the EMVD2 rule, we obtain

$$X_2(Y_2-Y_1) \text{ ->-> } Y_2 \cap Y_1|Z_2.$$

By the EMVD5 rule and the above two EMVDs, we conclude that

$$X_2(Y_2-Y_1) \text{ ->->} Y_1 \cap Y_2 | Z_1 Z_2$$

holds in R. Q.E.D.

Example 3.2: Assume that the following EMVDs hold in R=ABCDE:

AB ->-> CD|E,
C ->-> BD|A.

By Corollary 3.3, we obtain a new EMVD as follows:

BC ->-> D|AE.

Note that C ->-> BD holds in R'=ABCD, while C ->-> BD does not hold in R, but BC ->-> D holds in R=ABCDE.

3.3 Dependency Diagrams

In this section, we introduce a marked Hasse diagram to represent all the EMVDs that have a given set Z of attributes as their right side. This marked Hasse diagram, called a dependency diagram, is useful to investigate the interactions among EMVDs. Furthermore, we also consider whether or not these EMVDs hold when some attributes are added to or deleted from a relation scheme.

Beeri et al. introduced the concept of the 'dependency basis' of a given set X of attributes [BEERF7708]. Let $X, Y_1, \ldots, Y_k$ be disjoint sets of attributes. If $Y_1, \ldots, Y_k$ are the sets in the dependency basis of X, then any MVD, having X as the left side, can be represented by X ->-> Y, where Y is a

union of some attribute sets in the set $\{Y_1,\ldots,Y_k\}$. That is, the dependency basis of X contains all the information about MVDs that have X as their left side. When the dependency basis of X is a family $\{Y_1,\ldots,Y_k\}$, it is denoted by

$$X \twoheadrightarrow Y_1 | Y_2 | \ldots | Y_k.$$

Note that our concept of the dependency diagram is the reverse of the dependency basis since we consider a class of EMVDs in which one of the right side attribute set is fixed.

Definition 3.1: Let R and 2^R be a set of attributes and a set of all the subsets of R, respectively. Given a set Z of attributes such that $Z \subseteq R$, let H(Z) be a Hasse diagram of the partially ordered set $(2^{R-Z},>)$, where for any two elements X and Y in 2^{R-Z}, X>Y if and only if $X \subset Y$. A dependency diagram D(Z) has the same set of vertices and the same set of edges as H(Z). The interpretation of D(Z) is as follows: If two nodes <X> and <XY> are connected by an edge, then r[x,Z] r[xy,Z] holds for any XY-value xy in r[XY], and vice verse. Each edge connecting two vertices <X> and <XY> is called an equivalent edge if and only if r[x,Z]=r[xy,Z] holds for any XY-value xy in r[XY]. In D(Z), each equivalent edge is especially denoted by a double line.

Immediately from the above definition, we obtain the following theorem:

Theorem 3.3: In a dependency diagram D(Z), the vertices <X> and <XY> are connected by an equivalent edge if and only if the EMVD

X ->-> Y|Z holds in the relation r for D(Z), (proof omitted).

Hereafter, we use the dependency diagram to denote a set of EMVDs that hold in a given 'relation scheme'. That is, the EMVD X ->-> Y|Z holds in R if and only if the vertices <X> and <XY> are connected by an equivalent edge in the dependency diagram D(Z) for R.

Definition 3.2: In a dependency diagram D(Z), $(<X_i>,<X_j>)$ denotes an edge connecting the vertices $<X_i>$ and $<X_j>$. An edge sequence $(<X_1>,<X_2>),\ldots,(<X_{p-1}>,<X_p>)$ is defined as a sequence of edges such that for each i, $1\leq i\leq p-1$, $X_i>X_{i+1}$. An equivalent edge sequence is an edge sequence such that for each i, $1\leq i\leq p-1$, $(<X_i>,<X_{i+1}>)$ is an equivalent edge.

Hereafter, if we are interested in a certain family S of sets of attributes, each vertex in the dependency diagram is assumed to correspond to an element in 2^S. Obviously, the following properties are obtained by using the equivalent edge sequence.

Theorem 3.4: If both the vertices $<X_i>$ and $<X_j>$ $(X_i \subset X_j)$ appear in an equivalent edge sequence of D(Z). then an EMVD X_i ->-> X_j-X_i|Z holds in R (proof omitted).

Theorem 3.4 can be easily proved by using the EMVD5 rule repeatedly. Some equivalent edge sequences may imply the existence of other equivalent edge sequences as follows:

Theorem 3.5: If $(<X_1>,<X_2>),\ldots,(<X_{p-1}>,<X_p>)$ is an equivalent

edge sequence in D(Z), then any edge sequence in D(Z), whose starting vertex is $<X_1>$ and whose ending vertex is $<X_p>$, is also an equivalent edge sequence.

Example 3.3: Assume that X ->-> Y|Z and XY ->-> Z|W hold in R=WXYZ. The dependency diagram D(Z) of R is shown in Fig.3.2. Note that this dependency diagram illustrates the EMVD5 rule since (<X>,<XY>), (<XY>,<XYW>) is an equivalent edge sequence and thus, X ->-> Z|YW is proved to hold in R. Furthermore, the equivalent edge sequence implies the other equivalent edge sequence (<X>,<XW>),(<XW>.<XYW>). Therefore, X ->-> Y|Z and XY ->-> Z|W imply X ->-> W|Z and XW ->-> Y|Z. In this example, if we add the EMVD ϕ ->-> X|Z for R, then all the edges become equivalent edges from Theorem 3.5. That is, the EMVD ϕ ->-> XYW|Z becomes to hold in R.

Recently, Sagiv and Walecka generalized the concept of our dependency diagram and introduced the notion of subset dependencies [SAGIW7907]. By using similar diagram, they showed the following inference rule:

(EMVD6) [SAGIW7907]
If X ->-> Y|Z, Y ->-> W|Z and W ->-> X|Z hold in R, then Y ->-> X|Z holds in R.

This rule cannot be obtained by any combination of the rules EMVD0 - EMVD5. By using the existence of this rule, they also showed that there is no general finite set of inference rules for EMVDs. That is, the EMVD6 rule is easily generalized to n-ary EMVD rules [SAGIW7907]. Fig.3.3 illustrates the EMVD6 rule in our dependency diagram. In this diagram, only three

equivalent edges are shown such as (<X>,<XY>), (<Y>,<YW>) and (<W>,<XW>). However, we can verify that (<Y>.<XY>) is also an equivalent edge since

$$r[xy,Z] = r[x,Z] \supseteq r[xw,Z] =$$
$$r[w,Z] \supseteq r[yw,Z] = r[y,Z]$$

and $r[y,Z] \supseteq r[xy,Z]$ hold.

Here, we introduce the concept of the 'full' MVD and the left dependency basis.

D(Z)

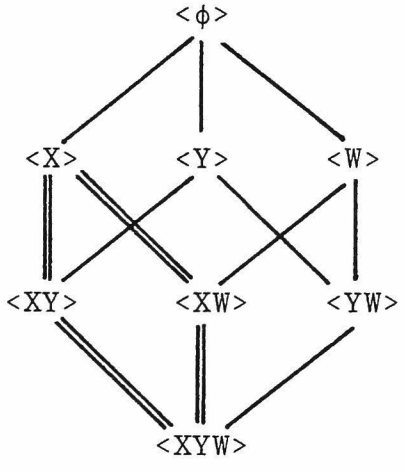


Fig.3.2. The dependency diagram D(Z) of R=XYZW with $\{X \twoheadrightarrow Z|Y,\ XY \twoheadrightarrow Z|W\}$.

D(Z)

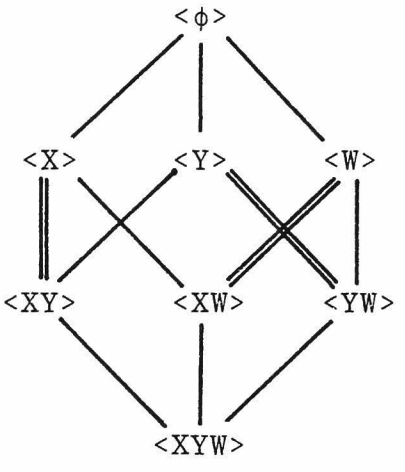


Fig.3.3. The dependency diagram to illustrate the EMVD6 rule.

Definition 3.3: If an MVD X ->-> Y holds in R such that for any proper subset X' of X, X' ->-> Y does not hold in R, then X ->-> Y is called a full MVD (for short, FMVD) and denoted by X ==>>Y. If the FMVD X_i ==>> Y, i=1,2,...,k, $X_i \cap Y = \phi$, hold in R, then we denote it by $X_1|X_2|...|X_k$ ==>> Y. If $X_1,...,X_k$ ($k \geq 1$) are all the sets of attributes such that X_i ==>> Y ($X_i \cap Y = \phi$) holds in R, then $\{X_1,...,X_k\}$ is called the left dependency basis of Y in R and we denote it by $X_1|X_2|...|X_k$ $\underset{L}{==>>}$ Y.

A left dependency basis $\{X_1,...,X_k\}$ of a given set Y in R is represented by the dependency diagram D(Y) in the following way. Each X_i corresponds to a starting vertex of an equivalent edge sequence that terminates at the vertex <R-Y>. Conversely, each starting vertex of any equivalent edge sequence that terminates at the vertex <R-Y> corresponds to some X_i.

The concept of the left dependency basis is useful to investigate whether a given set of MVDs hold after the addition or deletion of some attributes.

The major differences of the left dependency basis from the dependency basis are as follows:

(1) $X_1,...,X_k$ need not be disjoint sets of attributes.

(2) For any MVD X ->-> Y such that $X \cap Y = \phi$, X contains at least one X_i in the left dependency basis of Y.

Example 3.4: Assume that $A|B \underset{L}{==>>} C$ holds in R=ABCD. From the definition, we know that the left side of any MVD, having C as its right side, contains either A or B or both. Then, all the nontrivial MVDs X ->-> C such that $X \cap \{C\} = \phi$ are: A ->-> C, B ->-> C, AB ->-> C, AD ->-> C, BD ->-> C and ABD ->-> C. The dependency diagram D(C) of this relation scheme

R is shown in Fig.3.4.

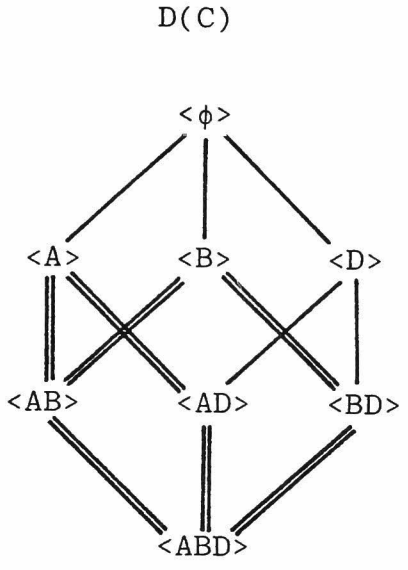


Fig.3.4. The dependency diagram D(C) of R=ABCD with $A|_{L}B$ ==>> C.

The following lemma shows that the minimality of the left side of any FMVD X ==>> Y is not influenced by deleting any attribute that does not belong to XY.

Lemma 3.1: If any FMVD X ==>> Y holds in R=XYZ, then X ==>> Y also holds in R'=XYZ', where Z ⊃ Z'.
(Proof). Assume that for some subset X' of X, X' ->-> Y holds in R'=XYZ'. That is, X' ->-> Y|Z'(X-X') holds in R'. By applying the EMVD4 rule to X' ->-> Y|Z'(X-X'), we obtain X' ->-> Y|X-X'. From the assumption X ->-> Y|Z, X'(X-X') ->-> Y|Z holds in R. By using the EMVD5 rule, we obtain X' ->-> Y|Z(X-X'). A contradiction. Q.E.D.

The following theorem shows that the minimality of any element in a left dependency basis is also invariant under the deletion of some attributes.

Theorem 3.6: Let R be a union of $X_1,\ldots,X_k,Y,Z$ such that $Z \cap (X_1 \ldots X_k Y) = \phi$. If $X_1\ X_2\ \ldots\ X_k \underset{L}{==>>} Y$ holds in R then $X_1|X_2|\ldots|X_k \underset{L}{==>>} Y$ also holds in $R'=X_1\ldots X_k YZ'$, where $Z \supset Z'$.
(Proof) See [TANAK7908].

3.4 Reducibility of Embedded Multivalued Dependencies

In this section, we consider how to store a given set of EMVDs in a reduced form. These discussions will be useful to handle the following problems:

(1) When a relational database scheme contains some relation schemes in which some nontrivial EMVDs hold, it is necessary to maintain those EMVDs whenever update operations are performed. If the EMVDs for a relation scheme is expressed in a reduced form, it is useful to decrease the cost for maintaining those EMVDs.

(2) As described in Section 3.1, some EMVDs on an initial relation scheme may not be reflected on the obtained set of relation schemes in Fagin's decomposition approach. In such a case, these EMVDs must be maintained as the 'interrelational dependencies' for the obtained set of relation schemes. Here, the interrelational dependencies mean the dependencies that are defined on a union of more than one relation schemes. In this case, it is also useful to store the unreflected EMVDs in a reduced form in order to decrease the cost for maintaining those EMVDs.

Definition 3.4: A set D of EMVDs is said to be redundant if and only if for some proper subset D' of D, D' implies D. If D is not redundant, then D is said to be nonredundant. Let X, Y and Z be disjoint sets of attributes. Assume that $X \supseteq X'$, $Y \supseteq Y'$, $Z \supseteq Z'$ and at least one of X', Y' and Z' is a proper subset. An EMVD d: X ->-> Y|Z in a given set D of EMVDs is said to be reducible to d': X' ->-> Y'|Z' within D if and only if D implies $(D-\{d\}) \cup \{d'\}$ and $(D-\{d\}) \cup \{d'\}$ implies D.

Theorem 3.7: Let D be a set of EMVDs X ->-> Y|Z and XY ->-> Z|W, where X, Y, Z and W are mutually disjoint sets of attributes. Then, D is nonredundant.
(Proof). It is sufficient to prove that X ->-> Y|Z does not imply XY ->-> Z|W and that XY ->-> Z|W does not imply X ->-> Y|Z. Fig.3.5(a) and (b) show such counterexample relations, respectively. Q.E.D.

(a)

X	Y	Z	W
x_1	y_1	z_1	w_1
x_1	y_2	z_1	w_1
x_1	y_1	z_2	w_2
x_1	y_2	z_2	w_2

(b)

X	Y	Z	W
x_1	y_1	z_1	w_1
x_1	y_1	z_2	w_1
x_1	y_1	z_1	w_2
x_1	y_1	z_2	w_2
x_1	y_2	z_2	w_1
x_1	y_2	z_2	w_2

Fig.3.5. Example relations on R=XYZW.

Theorem 3.8: The following nonredundant sets D_1 and D_2 are mutually implied. That is, D_1 implies D_2 and D_2 implies D_1.

$$D_1: \{X \text{ ->->} Y_1|Z,\ XY_1 \text{ ->->} Y_2Y_3|Z\}$$
$$D_2: \{X \text{ ->->} Y_2|Z,\ XY_2 \text{ ->->} Y_1Y_3|Z\}$$

Here, X, Y_1, Y_2, Y_3 and Z are mutually disjoint.
(Proof).
(1) (From D_1 to D_2) The EMVD5 rule results in $X \text{ ->->} Z|Y_1Y_2Y_3$ from D_1. By applying the EMVD4 rule to $X \text{ ->->} Z|Y_1Y_2Y_3$, we obtain the EMVD $X \text{ ->->} Y_2|Z$ in D_2. Furthermore, by applying the EMVD2 rule to $X \text{ ->->} Z|Y_1Y_2Y_3$, we obtain $XY_2 \text{ ->->} Y_1Y_3|Z$ in D_2.
(2) (From D_2 to D_1) This case is also proved in the same manner as the case of (1). Q.E.D.

Assume that an EMVD $X \text{ ->->} Y_1|Z_1$ holds in a relation scheme. In order that the EMVD $X \text{ ->->} Y_1Y_2|Z_1Z_2$ holds in R, at least the following EMVDS (shown in D_2 or D_3 except $X \text{ ->->} Y_1|Z_1$) must hold in R.

Theorem 3.9: The following sets D_1, D_2, D_3 of EMVDs are nonredundant, and D_1 implies D_2, D_2 implies D_3 and D_3 implies D_1.

$$D_1: \{X \text{ ->->} Y_1Y_2|Z_1Z_2\}$$
$$D_2: \{X \text{ ->->} Y_1|Z_1,\ XY_1 \text{ ->->} Y_2|Z_1,\ XZ_1 \text{ ->->} Y_1Y_2|Z_2\}$$
$$D_3: \{X \text{ ->->} Y_1|Z_1,\ XZ_1 \text{ ->->} Y_1|Z_2,\ XY_1 \text{ ->->} Y_2|Z_1Z_2\}.$$

Here, X, Y_1, Y_2, Z_1 and Z_2 are disjoint sets of attributes.
(Proof).

(1) (From D_1 to D_2) By the EMVD4 rule, we obtain the EMVD $X \rightarrow\!\!\!\!\rightarrow Y_1 | Z_1$ from D_1. By applying the EMVD2 rule and the EMVD4 rule to $X \rightarrow\!\!\!\!\rightarrow Y_1Y_2 | Z_1Z_2$, we obtain $XY_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_1$ and $XZ_1 \rightarrow\!\!\!\!\rightarrow Y_1Y_2 | Z_2$.

(2) (From D_2 to D_3) By applying the EMVD2 rule to $XZ_1 \rightarrow\!\!\!\!\rightarrow Y_1Y_2 | Z_2$ in D_2, we obtain $XY_1Z_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_2$. By the EMVD5 rule, $XY_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_1Y_2$ in D_3 is derived from both $XY_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_1$ and $XY_1Z_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_2$. By the EMVD4 rule, $XZ_1 \rightarrow\!\!\!\!\rightarrow Y_1 | Z_2$ in D_3 is derived from $XZ_1 \rightarrow\!\!\!\!\rightarrow Y_1Y_2 | Z_2$ in D_2.

(3) (From D_3 to D_1) By the EMVD5 rule, $X \rightarrow\!\!\!\!\rightarrow Y_1 | Z_1Z_2$ is derived from both $X \rightarrow\!\!\!\!\rightarrow Y_1 | Z_1$ and $XZ_1 \rightarrow\!\!\!\!\rightarrow Y_1 | Z_2$. Furthermore, by the EMVD5 rule, D_1 is obtained from $XY_1 \rightarrow\!\!\!\!\rightarrow Y_2 | Z_1Z_2$ in D_3 and $X \rightarrow\!\!\!\!\rightarrow Y_1 | Z_1Z_2$. From the definition of the redundancy, D_1 is obviously nonredundant. Fig.3.6 (a), (b) and (c) show the example relations in which

(a)

X	Y_1	Y_2	Z_1	Z_2
1	1	2	1	2
1	3	2	3	2
1	1	2	1	4
1	3	2	3	4

(b)

X	Y_1	Y_2	Z_1	Z_2
1	1	2	1	2
1	3	2	1	2
1	1	4	3	4
1	3	4	3	4

(c)

X	Y_1	Y_2	Z_1	Z_2
1	1	2	1	2
1	3	2	1	3
1	1	2	4	2
1	3	2	4	3

Fig.3.6. Example relations on $R=XY_1Y_2Z_1Z_2$.

$$D_2-\{X \text{ ->->} Y_1|Z_1\} \not\models D_2,$$
$$D_2-\{XY_1 \text{ ->->} Y_2|Z_1\} \not\models D_2,$$
$$D_2-\{XZ_1 \text{ ->->} Y_1Y_2|Z_2\} \not\models D_2 ,$$

respectively. Therefore, D_2 is nonredundant. In the same way, D_3 is proved to be nonredundant. Q.E.D.

From Theorem 3.9, we immediately obtain the following theorem concerned with the reducibility of EMVDs:

Theorem 3.10: Let X, Y_1, Y_2, Z_1 and Z_2 be disjoint sets of attributes. An EMVD $X \text{ ->->} Y_1Y_2|Z_1Z_2$ in a given set D of EMVDs is reducible to $X \text{ ->->} Y_1|Z_1$ within D if and only if

$$(D-\{X \text{ ->->} Y_1Y_2|Z_1Z_2\}) \cup \{X \text{ ->->} Y_1|Z_1\}$$

implies $XY_1 \text{ ->->} Y_2|Z_1$ and $XZ_1 \text{ ->->} Y_1Y_2|Z_2$ (proof omitted).

In the following theorem, we provide a sufficient condition such that an EMVD $X_1X_2 \text{ ->->} Y_1Y_2|Z_1Z_2$ in a given set D is reducible to $X_1 \text{ ->->} Y_1|Z_1$ within D:

Theorem 3.11: Let d, d' and d'' denote the EMVDs $X_1X_2 \text{ ->->} Y_1Y_2|Z_1Z_2$, $X_1X_2 \text{ ->->} Y_1|Z_1$ and $X_1 \text{ ->->} Y_1|Z_1$, respectively. Let D_1 be a set of EMVDs that contain d, and let D_2 be a set $(D_1-\{d\}) \cup \{d''\}$. Then, d is reducible to d'' within D_1, that is, D_1 and D_2 are mutually implied if
(1) d is reducible to d' within D_1,
(2) $(D_1-\{d\}) \cup \{d'\}$ implies $X_1 \text{ ->->} X_2|Y_1$, and
(3) D_2 implies $X_1Z_1 \text{ ->->} Y_1|X_2$.
(Proof). From (1), $(D_1-\{d\}) \cup \{d'\}$ implies D_1 and D_1 also

implies $(D_1-\{d\}) \cup \{d'\}$. By Theorem 3.8, the following two nonredundant sets are mutually implied:

D_3: $\{X_1 \text{ ->-> } X_2|Y_1,\ d'\}$
D_4: $\{d'',\ X_1Z_1 \text{ ->-> } Y_1|X_2\}$.

From (2), $(D_1-\{d\}) \cup \{d'\}$ implies D_3 and from (3), D_2 implies D_4. $(D_1-\{d\}) \cup \{d'\}$ implies D_2 since D_3 implies d''. Furthermore, D_2 implies $(D_1-\{d\}) \cup \{d'\}$ since D_4 implies d'. Therefore, D_1 and D_2 are mutually implied. Q.E.D.

Example 3.5: Let D_1 be a set of the following EMVDs:

AB ->-> C|DE,
AB ->-> D|CE,
A ->-> BE|D.

Let us consider the reducibility of the EMVD AB ->-> D|CE within D_1. From Theorem 3.10, AB ->-> D|CE is reducible to AB ->-> D|E since AB ->-> D|E and ABE ->-> D|C (implied by AB ->-> C|DE in D_1) imply the original EMVD AB ->-> D|CE. Furthermore, AB ->-> D|CE is reducible to A ->-> D|E within D_1 since
(1) AB ->-> D|CE is reducible to AB ->-> D|E in D_1,
(2) A ->-> B|D (i.e., X_1 ->-> $X_2|Y_1$ in Theorem 3.11) is implied by A ->-> BE|D, and
(3) AE ->-> D|B (i.e., X_1Z_1 ->-> $Y_1|X_2$ in Theorem 3.11) is implied by A ->-> BE|D.
Therefore, D_1 and the following D_2 are mutually implied:

D_2: $\{A \text{ ->-> } D|E,$

AB ->-> C|DE,
A ->-> BE|D}.

Furthermore, D_2 is redundant since A ->-> D|E in D_2 is obtained by applying the EMVD2 rule to A ->-> BE|D in D_2. Therefore, D_2 is equivalent to the following set D_3:

D_3: { AB ->-> C|DE,
A ->-> BE|D}.

3.5 Concluding Remarks

In this chapter, we have shown several properties of EMVDs. From the motivation stated in Section 3.1, we investigate several inference rules for EMVDs and for the interaction between MVDs and EMVDs. The equivalence testing problem described in Section 3.1 will be further discussed in the succeeding section. We emphasize that even if we design a relational database scheme by 'MVDs', it is necessary to handle the EMVDs in order to solve the equivalence testing problem sufficiently.

As described in Section 3.2, our basic rule for EMVDs is recently called the joinability rule by Parker et al. This is because the EMVD5 rule and Corollary 3.1 are strongly related to the join operation to preserve some MVD after the addition of some attributes.

The concept of the dependency diagram was introduced in this chapter. The dependency diagram is useful to investigate the interactions among EMVDs. Furthermore, the notion of the dependency diagram is recently generalized by Sagiv et al. as

described in Section 3.3 in order to find an alternative of EMVDs.

Recently, Sagiv and Walecka [SAGIW7907] and independently, Parker and Parsaye-Ghomi [PARKP8005] showed that there does not exist a general finite complete set of inference rules for EMVDs. Some example was shown by our dependency diagram in Section 3.3. Ito et al. [ITO-T8009] showed a set of inference rules for FDs, MVDs and EMVDs to derive all the EMVDs in a restricted case, where some EMVDs are given for only a certain subset W of attributes of a relation scheme and the derivable EMVDs are only those which have a subset of W as their left side. Biskup [BISK8001] showed a set of inference rules for MVDs in undetermined universes and discussed the semantic aspects of a class of MVDs that are derivable from these inference rules. Biskup's approach is important and is strongly related to the study of EMVDs since MVDs in an undetermined universe mean the MVDs that do not vary after additions or deletions of attributes. Further research will be needed for the following problems:

(1) Not all EMVDs are useful to represent semantic element-set correspondence between sets of attributes in a real world description. It will be important to find a complete set of inference rules for EMVDs when the EMVDs are restricted to only those which represent a 'real world' description.

(2) We showed that some EMVDs can be expressed in a reduced form in Section 3.4. It will be necessary to find an algorithm to transform a given set of EMVDs into a set of EMVDs whose number of attributes is minimized.

CHAPTER 4 PRESERVATIONS OF DATA DEPENDENCIES

In this chapter, we introduce the notion of the 'preservation' of data dependencies. We mainly consider what data dependencies can be preserved by a set of relation schemes.

The major difference of our results from others is that our results are obtained without the 'universal relation assumption'. This assumption is impractical since it enforces that all the relations must be projections of a common large relation at every time. We allow each relation to be updated independently from others. Under this environment, we provide conditions for FDs, EMVDs and EJDs to be preserved by a set of relation schemes.

4.1 Introduction

A central problem of the relational database design theory is how to select a 'better' relational database scheme (a set of relation schemes), that is 'equivalent' to a given relational database scheme [BEERB7809]. One aspect of the 'equivalence' is the equivalence between two sets of constraints enforced by two relational database schemes. When the constraints are only those which are represented as data dependencies, it is necessary to test whether one relational database scheme can enforce the same data dependencies as the other.

In this chapter, we consider what data dependencies can be enforced by a relational database scheme by the notion of the 'preservation' of data dependencies. Several concepts related to the 'equivalence' of relational database schemes are known [RISS7712], [BEERM7904], [AHO-B7909], [MAIEM7912], [BEERB8001]. The major difference of our 'preservation' from these concepts

is that we do not have the 'universal relation assumption', which enforces that all relations must be projections of a large common relation at every time.

We believe that the universal relation assumption is too strict since it leads to the introduction of more interrelational constraints: If an attribute A belongs to two relation schemes R_1 and R_2, then this assumption enforces that the two sets of A-values in relations r_1 and r_2 should be the same at any time. Even if we have a tuple t to be inserted into r_1, we cannot insert it until we have some tuple of r_2, whose A-value is the same as the A-value of t. This may cause further insertion/deletion anomalies shown in [DATE77]. Furthermore, recently, Honeyman et al. showed that the problem of testing and maintaining the universal relation assumption are NP-complete. We consider conditions that a data dependency is enforced by a relational database scheme when a relation on each relation scheme is allowed to be updated independently from others.

Probably, the most natural way to test whether a data dependency is preserved by a relational database $\{r_1,\ldots,r_n\}$ is to take the join of the r_i's, and to test whether the join of the r_i's satisfies the data dependency.

When a set of FDs holds in a relation r, any relation, that is a join of r and any other relations, also satisfies the same set of FDs. In this sense, any FD defined on a relation scheme can be preserved.

For EMVDs (containing MVDs), there are two serious problems as follows:

(1) Some relational database scheme $\{R_1,\ldots,R_n\}$ can enforce an EMVD on the union of R_i's, that is not defined on any R_i as a nontrivial EMVD.

(2) An EMVD, defined on some R_i, can not be necessarily enforced

on the union of the R_i's.

As for (1), let R_1=AB and R_2=AC be two relation schemes, where any nontrivial data dependency does not hold in R_1 and in R_2. Fig.4.1(a) shows such example relations r_1 on R_1 and r_2 on R_2. If we take the join r_1*r_2 as shown in Fig.4.1(b). then the join r_1*r_2 always satisfies the EMVD A ->-> B|C. We should note that the EMVD A ->-> B|C can not be defined on R_1 nor R_2. For this problem, Maier et al. [MAIEM7912] and Beeri et al. [BEERB8001] have already provided useful formalisms.

(a)

A	B
0	0
0	1
1	1

r_1

A	C
0	1
0	0
2	1

r_2

(b)

A	B	C
0	0	1
0	1	1
0	0	0
0	1	0

r_1*r_2

Fig.4.1. Example relations r_1, r_2 and the join r_1*r_2.

The problem (2) is serious if we do not have the universal relation assumption. For example, let R_1=ABCD and R_2=BCE be two relation schemes, where the EMVD A ->-> B|CD holds in R_1 and the FD BC -> E holds in R_2. Fig.4.2(a) shows example relations r_1 on R_1 and r_2 on R_2 such that r_1 satisfies the EMVD A ->-> B|CD and r_2 satisfies BC -> E. If we take a join r_1*r_2 as shown in Fig.4.2(b), then the EMVD A ->-> B|CD does not hold in r_1*r_2. This is because the fourth tuple (1,1,0,1) in r_1 does not appear

(a) r_1

A	B	C	D
1	0	1	0
1	0	0	1
1	1	1	0
1	1	0	1

(A ↠ B|CD)

r_2

B	C	E
0	1	0
0	0	0
1	1	0

(BC → E)

(b) $r_1 * r_2$

A	B	C	D	E
1	0	1	0	0
1	0	0	1	0
1	1	1	0	0

(BC → E holds. A ↠ B|CD does not hold.)

Fig.4.2. Preservability of FDs and EMVDs.

in $r_1 * r_2$ since there is not any tuple t whose B-value is '1' and whose C-value is '0'. Suppose that a relation scheme $R=R_1 \cup R_2$ satisfies the set of dependencies {A ->-> B|CD, CD -> E}. Then this example shows that the relational database scheme $\{R_1, R_2\}$ cannot enforce the same data dependencies as the relational database scheme {R}. This problem is not dealt with in [MAIEM7912] nor [BEERB8001].

In this chapter, first, we provide precise definitions of the preservations of FDs, EMVDs and EJDs when each relation is allowed to be updated independently from others. Several related concepts will be also comared with our 'preservation' of data dependencies. In Section 4.3, we consider only the preservations of data dependencies, which are defined on at least one given relation scheme, by a relational database scheme. In Section 4.4, we introduce the notion of the Dependency Preserving Normal Form (for short, DPNF) of a relation scheme. Under the assumption that each relation scheme

R_i is in DPNF, we provide a necessary and sufficient condition such that a data dependency (not necessarily defined on some R_i as a nontrivial dependency) is preserved by the relational database scheme $\{R_1,\ldots,R_n\}$. In Section 4.5, we discuss conditions to test whether a relational database scheme consisting of not necessarily DPNF relation schemes can preserve a data dependency.

4.2 Basic Concepts

In this section, we provide formal definitions of the preservations of several kinds of data dependencies. We also show several concepts related to the notion of our 'preservation' and clarify the differences among them.

Definition 4.1: Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme such that $R=\bigcup_{i=1}^{n} R_i$. Let D_i denote a given set of nontrivial data dependencies (FDs, EMVDs and EJDs) that hold in R_i for each i, $1\leq i\leq n$. The preservations of FDs, EMVDs and EJDs are defined as follows:

(1) Preservation of FDs: Let X -> Y be an arbitrary FD. We say $\{R_1,\ldots,R_n\}$ preserves X -> Y if and only if (a) $XY \subseteq R$ and (b) the join of the r_i's, $*_{i=1}^{n} r_i$ satisfies X -> Y for any database $\{r_1,\ldots,r_n\}$ such that r_i belongs to $SAT(D_i)$ for each i, $1\leq i\leq n$.

(2) Preservation of EMVDs: Let X ->-> Y|Z be an arbitrary EMVD. We say $\{R_1,\ldots,R_n\}$ preserves the EMVD X ->-> Y|Z if and only if (a) $XYZ \subseteq R$ and (b) $*_{i=1}^{n} r_i$ satisfies X ->-> Y|Z for any database

$r_1,\ldots,r_n$ such that r_i belongs to $SAT(D_i)$ for each i, $1<i<n$.
(3) Preservation of EJDs: Let $*[S_1,\ldots,S_m]$ be an arbitrary EJD. We say $\{R_1,\ldots,R_n\}$ preserves the EJD $*[S_1,\ldots,S_m]$ if and only if (a) $R \supseteq \overset{m}{\underset{i=1}{U}} S_i$ and (b) $\overset{n}{\underset{i=1}{*}} r_i$ satisfies $*[S_1,\ldots,S_m]$ for any database $\{r_1,\ldots,r_n\}$ such that r_i belongs to $SAT(D_i)$ for each i, $1\leq i\leq n$.
Here, $SAT(D_i)$ denotes a set of all the relations on R_i that satisfy every dependency in D_i.

Let d be an arbitrary dependency (FD, EMVD or EJD) that can be defined on $R=\overset{n}{\underset{i=1}{U}} R_i$. Assume that SAT(d) denotes a set of all the relations on R that satisfy d. From the definitions above, the relational database scheme $\{R_1,\ldots,R_n\}$ preserves d if and only if

$$\overset{n}{\underset{i=1}{*}} r_i \in SAT(d)$$

for any database $\{r_1,\ldots,r_n\}$ such that r_i belongs to $SAT(D_i)$ for each i, $1\leq i\leq n$. Fig.4.3 illustrates the concept of the preservation of a data dependency d.

If $\{R_1,\ldots,R_n\}$ preserves d, then we can define a mapping P such that

$$P:\ SAT(D_1)\ x\ldots x\ SAT(D_n) \longrightarrow SAT(d),$$
$$(r_1,\ldots,r_n) \longmapsto \overset{n}{\underset{i=1}{*}} r_i.$$

Note that P is a mapping from $SAT(D_1)x\ldots xSAT(D_n)$ 'into' SAT(d), and not necessarily an 'onto' mapping. Furthermore, P is not necessarily a one-to-one correspondence between $SAT(D_1)x\ldots xSAT(D_n)$ and

$$\{ \overset{n}{\underset{i=1}{*}} r_i \mid \text{for each } i,\ r_i \in SAT(D_i) \}.$$

That is, for some different lists of relations $(r_1,\ldots,r_n)$ and $(r_1',\ldots,r_n')$ in $SAT(D_1)x\ldots xSAT(D_n)$.

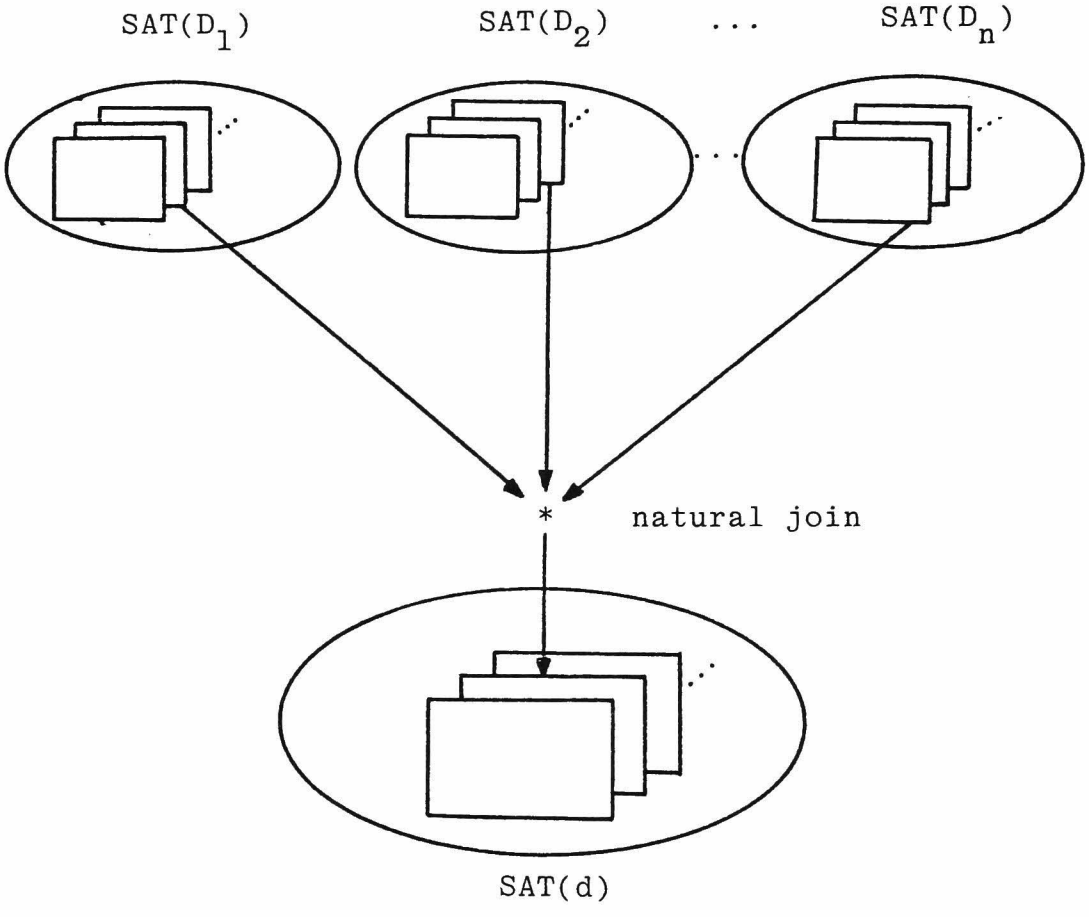


Fig.4.3. The preservation of a data dependency d by a relational database scheme $\{R_1,\ldots,R_n\}$.

$$\overset{n}{\underset{i=1}{*}} r_i = \overset{n}{\underset{i=1}{*}} r_i'$$

may hold. Fig.4.4 illustrates the mapping P defined by the fact that $\{R_1,\ldots,R_n\}$ preserves d.

The following concepts are well known as useful criteria for the equivalence between two relational database schemes:

(1) Lossless join Property [AHO-B7909]

(2) Independent components and Faithful representation [RISS7712] [BEERR8001]

(3) 'Preservation of data dependencies' in [BEERM7904]

(4) Adequacy for decompositions [MAIEM7912]

The major differences of our notion of 'preservation' from them are as follows:

(a) Our notion of 'preservation' is concerned with only the equivalence between sets of dependency constraints enforced by two relational database schemes.

(b) We do not have the universal relation assumption as described in Section 4.1.

The lossless join property assumes the existence of a 'universal' relation scheme R, which consists of all the attributes, and a set D of data dependencies on R. It guarantees only the fact that it is possible to reconstruct any universal relation r on R that satisfy D by joining its projections in the following way:

Definition 4.2: A relational database scheme $\{R_1,\ldots,R_n\}$ is said to have the lossless join property if and only if, for all relations r on $R=\overset{n}{\underset{i=1}{U}} R_i$ that satisfy D,

$$r = \overset{n}{\underset{i=1}{*}} r[R_i].$$

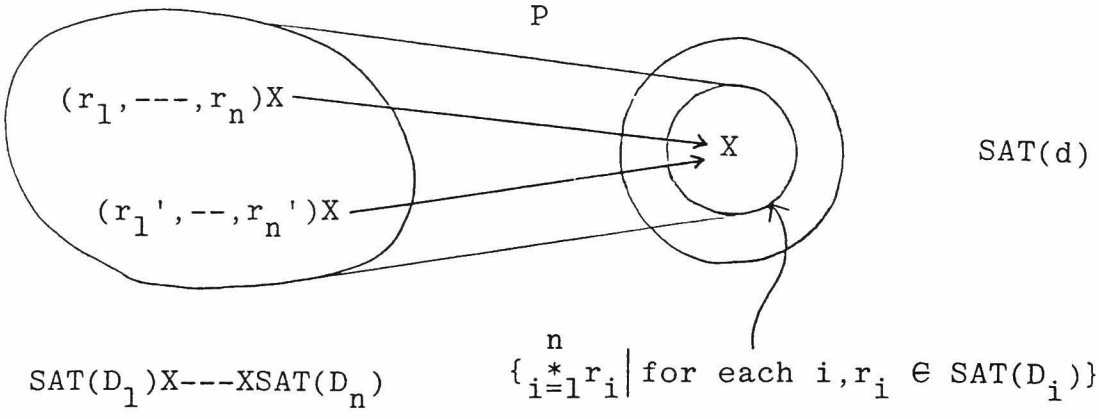


Fig.4.4. The mapping P defined by the fact that $\{R_1,---,R_n\}$ preserves d.

(a)

A	B	C	D
0	0	0	0
0	0	1	0
1	0	0	0

r

A	B
0	0
1	0
2	1

r_1

B	C	D
0	0	0
0	1	0
1	1	1
2	1	1

r_2

(b)

A	B	C	D
0	0	0	0
0	0	1	0
1	0	0	0
1	0	1	0

$r[R_1]*r[R_2]$

(c)

A	B	C	D
0	0	0	0
0	0	1	0
1	0	0	0
1	0	1	0
2	1	1	1

r_1*r_2

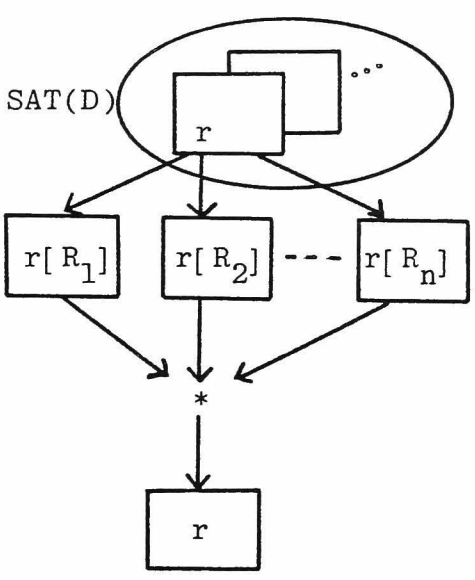
Fig.4.5. Example relations for lossless join property and preservation of data dependencies.

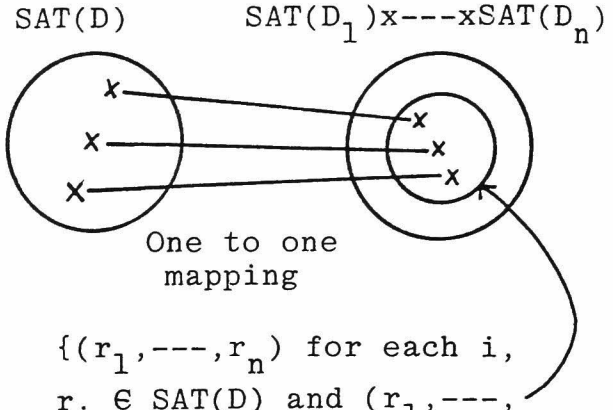
Here, R is a 'universal' relation scheme, and D is a given set of data dependencies that hold in R.

In this sense, the lossless join property is strongly related to the 'data equivalence' [BEERM7904] between two relational database schemes. The lossless join property does not imply the preservation of dependencies, and the preservation of dependencies does not also imply the lossless join property. Consider the relation schemes R=ABCD, R_1=AB and R_2=BCD with D= A -> B, BC -> D , D_1= A -> B and D_2= BC -> D , respectively. Fig.4.5(a) shows the example relations r on R, r_1 on R_1 and r_2 on R_2 which satisfy D, D_1 and D_2, respectively. As shown in Fig.4.5(b), the join $r[R_1]*r[R_2]$ is not equal to r, and therefore $\{R_1,R_2\}$ does not satisfy the lossless join property. We can, however, verify that r_1*r_2 (shown in Fig.4.5(c)) satisfies every dependency in D. In fact, from the discussions in the following sections, we can prove that $\{R_1,R_2\}$ preserves every dependency in D. Fig.4.6(a) illustrates the concept of the lossless join property.

The concept of 'independent components' originally dealt with only FDs. If $R_1,\ldots,R_n$ are independent components, then the lossless join property is satisfied by the $\{R_1,\ldots,R_n\}$ and the given set F of FDs on $R=\bigcup_{i=1}^{n} R_i$ is implied by the union of F_i's, where each F_i denotes a set of FDs on R_i that are implied by F. Recently, the concept was generalized into the concept of 'faithful representation' [BEERR8001] and 'adequacy of decomposition' [MAIEM7912] in order to handle JDs on R as well as FDs on R. Their conditions are as follows:
(i) the obtained relational database scheme $\{R_1,\ldots,R_n\}$ satisfies the lossless join property for R and D, and
(ii) the given set D of dependencies on R is implied by the

(a) Lossless join property (b) Faithful representation





{(r_1,---,r_n) for each i, r_i ∈ SAT(D) and (r_1,---, r_n)is a list of projections of a relation on R}

(c) "Preservation of data dependencies" by Beeri et al.

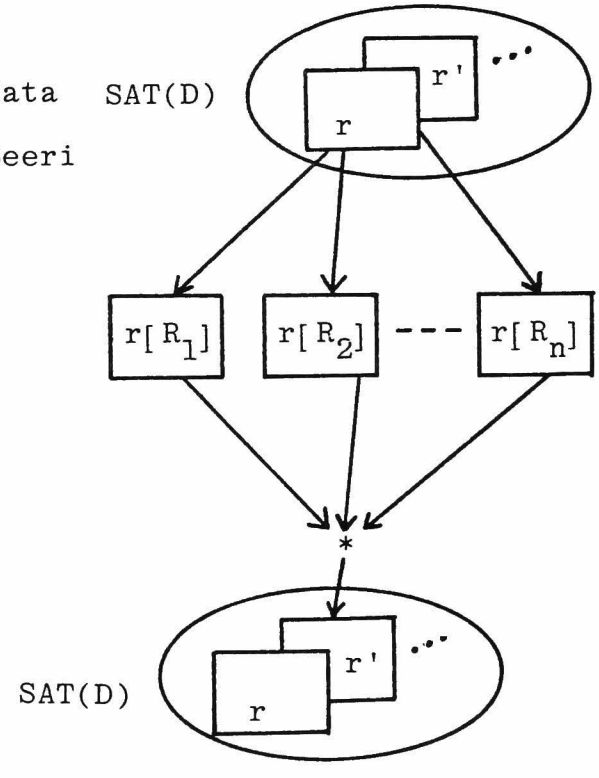


Fig.4.6. Several criteria for equivalence of relational database schemes.

union of D_i and the JD $*[R_1,\ldots,R_n]$, where D_i is a set of FDs and JDs on R_i that are implied by D.
These conditions seem to be similar to our results obtained independently in Section 4.4. However, the differences are that we do not have the universal relation assumption but that they use the assumption. Furthermore, we do not assume even the existence of the 'universal' relation scheme. Fig.4.6(b) illustrates the concept of independennt components and faithful representation.

Beeri et al. discussed the preservation of data dependencies under the universal relation assumption. As shown in Fig.4.6(c), they examined whether the join of projections $r[R_i]$'s belong to SAT(D) for any relation r on R that satisfies D. This concept is a relaxation of the lossless join property, and it also uses the universal relation assumption.

4.3 Preserving Data Dependencies That Hold in a Relation Scheme

As shown in Section 4.1, there are two types of data dependencies that are preserved by a relational database scheme. One is a class of data dependencies that originally hold in at least one relation scheme of the relational database scheme. The other is a class of data dependencies that are produced by taking the join of r_i's in the database.

In this section, we consider the preservation of data dependencies that hold in at least one relation scheme.

The following theorem shows that any FD, that holds in a relation scheme R_i, is preserved by any relational database scheme containing R_i.

Theorem 4.1: [TANAK8002]
Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme such that the FD X -> Y holds in some R_i. Then, $\{R_1,\ldots,R_n\}$ always preserves the FD X -> Y.
(Proof). Assume that the FD X -> Y holds in R_1. Suppose that the FD X -> Y is not satisfied by some $r=\overset{n}{\underset{i=1}{*}}r_i$, where r_1 is a relation on R_1 that satisfies X -> Y. There must exist two distinct tuples t_1 and t_2 in r such that $t_1[X]=t_2[X]$ and $t_1[Y]\neq t_2[Y]$. Since r contains tuples t_1 and t_2, r_1 must contain tuples t_1' and t_2' such that $t_1'[X]=t_2'[X]$ and $t_1'[Y]\neq t_2'[Y]$. This leads to the fact that r_1 does not satisfy the FD X -> Y. A contradiction. Q.E.D.

Let us consider the preservation of an MVD X ->-> Y, which holds in a relation scheme R_i. When a relational database scheme $\{R_1,\ldots,R_n\}$ contains R_i, generally, $\overset{n}{\underset{j=1}{\cup}}R_j \supseteq R_i$. Therefore, we consider whether the EMVD X ->-> $Y|R_i-XY$ is preserved by the relational database scheme. The following theorem provides a necessary and sufficient condition for the EMVD X ->-> $Y|R_i-XY$ to be preserved by a relational database scheme consisting of only two relation schemes.

Theorem 4.2: Let $R_1=XYZ$ and R_2 be relation schemes, where X, Y and Z are mutually disjoint. Assume that the EMVD X ->-> Y|Z holds in R_1. The EMVD X ->-> Y|Z is preserved by $\{R_1,R_2\}$ if and only if the EMVD X ->-> $Y\cap R_2|Z\cap R_2$ is preserved by $\{R_1,R_2\}$.
(Proof). The proof of the 'only if' part is straightforward since X ->-> Y|Z implies X ->-> $Y\cap R_2|Z\cap R_2$.
Without loss of generality, we can assume that $R_1=X_1X_2Y_1Y_2Z_1Z_2$, $R_2=WX_2Y_2Z_2$, $X=X_1X_2$, $Y=Y_1Y_2$ and $Z=Z_1Z_2$, where W, X_1, X_2, Y_1, Y_2, Z_1 and Z_2 are mutually disjoint sets of attributes. Suppose

that the EMVD X ->-> Y|Z (that is, X_1X_2 ->-> $Y_1Y_2|Z_1Z_2$) is not preserved by $\{R_1,R_2\}$. Then, there must exist some relations r_1 on R_1 and r_2 on R_2 such that r_1 satisfies X ->-> Y|Z, but r_1*r_2 does not satisfy X ->-> Y|Z. Let r be a projection of r_1*r_2 onto XYZ, that is, $r=(r_1*r_2)[XYZ]$. In r, there must exist the tuples

$$t_1: (x_1,x_2,y_1,y_2,z_1,z_2),$$
$$t_2: (x_1,x_2,y_1',y_2',z_1',z_2')$$

such that the following tuple t_3

$$t_3: (x_1,x_2,y_1,y_2,z_1',z_2')$$

is not contained in r. Since $\{R_1,R_2\}$ preserves X ->-> $Y \cap R_2|Z \cap R_2$, that is, X_1X_2 ->-> $Y_2|Z_2$, if $r[X_1X_2Y_2Z_2]$ contains the tuples

$$t_1': (x_1,x_2,y_2,z_2),$$
$$t_2': (x_1,x_2,y_2',z_2'),$$

then $r[X_1X_2Y_2Z_2]$ must also contain the tuple t_3'

$$t_3': (x_1,x_2,y_2,z_2')$$

If r contains the tuples t_1 and t_2, then $r[X_1X_2Y_2Z_2]$ must contain t_1', t_2' and t_3'. Since r_1 contains tuples t_1, t_2 and t_3, and $r_2[X_2Y_2Z_2]$ contains tuples (x_2,y_2,z_2), (x_2,y_2',z_2') and (x_2,y_2,z_2'), r must contain the tuple t_3. A contradiction. Q.E.D.

Theorem 4.2 provides a necessary and sufficient condition

in such a form that another EMVD (that is, $X \twoheadrightarrow Y \cap R_2 | Z \cap R_2$) must be also preserved by $\{R_1, R_2\}$. When some dependencies hold in R_1, R_2 or both, the condition shown in Theorem 4.2 automatically holds. The following theorem provides a sufficient condition to test by checking the dependencies of R_1 and R_2 whether the condition in Theorem 4.2 holds. Hereafter, we assume that for any relational database scheme, whenever a data dependency holds in a relation scheme, it also holds in any other relation schemes on which the dependency can be defined.

Theorem 4.3: [TANAK7904][KAMBT7911R][TANAK8002]
Let $\{R_1, R_2\}$ be an arbitrary relational database scheme such that $R_1=XYZ$ and X, Y and Z are mutually disjoint. Assume that $X \twoheadrightarrow Y|Z$ holds in R_1. The EMVD $X \twoheadrightarrow Y|Z$ is preserved by $\{R_1, R_2\}$ if
(1) either the FD $X \rightarrow Y \cap R_2$ or $X \rightarrow Z \cap R_2$ holds in R_1, or
(2) the EMVD $X \cap R_2 \twoheadrightarrow Y \cap R_2 | Z \cap R_2$ holds in R_2.
(Proof). Without loss of generality, we assume that $R_1=X_1X_2Y_1Y_2Z_1Z_2$ and $R_2=WX_2Y_2Z_2$, where W, X_1, X_2, Y_1, Y_2, Z_1 and Z_2 are mutually disjoint. If we regard X, Y and Z as X_1X_2, Y_1Y_2 and Z_1Z_2, respectively, then it is sufficient to show that $\{R_1, R_2\}$ preserves the EMVD $X_1X_2 \twoheadrightarrow Y_1Y_2|Z_1Z_2$.
(Case 1). Suppose that $X \rightarrow Y \cap R_2$, that is $X_1X_2 \rightarrow Y_2$ holds in R_1. From Theorem 4.1, the FD $X_1X_2 \rightarrow Y_2$ is preserved by $\{R_1, R_2\}$ since $X_1X_2Y_2 \subseteq R_1$. Therefore, the EMVD $X_1X_2 \twoheadrightarrow Y_2|Y_1Z_1Z_2W$ is preserved by $\{R_1, R_2\}$. From this and the projection rule for EMVDs, $\{R_1, R_2\}$ also preserves the EMVD $X_1X_2 \twoheadrightarrow Y_2|Z_2$. By Theorem 4.2, the EMVD $X_1X_2 \twoheadrightarrow Y_1Y_2|Z_1Z_2$ is proved to be preserved by $\{R_1, R_2\}$ since $X \twoheadrightarrow Y \cap R_2 | Z \cap R_2$ is equal to $X_1X_2 \twoheadrightarrow Y_2|Z_2$.

(Case 2). Suppose that $X \cap R_2 \text{ ->-> } Y \cap R_2 | Z \cap R_2$, that is, $X_2 \text{ ->-> } Y_2|Z_2$ holds in R_2. Let r_1 be an arbitrary relation on R_1 that satisfies $X_1X_2 \text{ ->-> } Y_1Y_2\ Z_1Z_2$. Let r_2 be an arbitrary relation on R_2 that satisfies $X_2 \text{ ->-> } Y_2|Z_2$. Consider an arbitrary tuple t in $r=r_1*r_2$ such that $t(X_1X_2)=x_1x_2$. From the projection rule for EMVDs, r_1 satisfies the EMVD $X_1X_2 \text{ ->-> } Y_2|Z_2$. Consequently,

$$r_1[x_1x_2, Y_2Z_2] = r_1[x_1x_2, Y_2] \times r_1[x_1x_2, Z_2].$$

From the assumption that r_2 satisfies $X_2 \text{ ->-> } Y_2|Z_2$,

$$r_2[x_2, Y_2Z_2] = r_2[x_2, Y_2] \times r_2[x_2, Z_2].$$

In $r=r_1*r_2$,

$$\begin{aligned} r[x_1x_2, Y_2Z_2] &= r_1[x_1x_2, Y_2Z_2] \cap r_2[x_2, Y_2Z_2] \\ &\quad (r_1[x_1x_2, Y_2] \times r_1[x_1x_2, Z_2]) \cap \\ &\quad (r_2[x_2, Y_2] \times r_2[x_2, Z_2]) \\ &= (r_1[x_1x_2, Y_2] \cap r_2[x_2, Y_2]) \\ &\quad \times (r_2[x_1x_2, Z_2] \cap r_2[x_2, Z_2]) \\ &= r[x_1x_2, Y_2] \times r[x_1x_2, Z_2]. \end{aligned}$$

Therefore, r_1*r_2 satisfies $X_1X_2 \text{ ->-> } Y_2|Z_2$. From Theorem 4.2, we conclude that R_1, R_2 preserves the EMVD $X_1X_2 \text{ ->-> } Y_1Y_2|Z_1Z_2$. Q.E.D.

Example 4.1: Let R_1 and R_2 be relation schemes such that R_1=BCDEFG and R_2=ACEG. Assume that the EMVD BC ->-> DE|FG holds in R_1. Then, from Theorem 4.3, if

(1) the FD BC -> E ($X \rightarrow Y \cap R_2$ in Theorem 4.3) holds in R_1, or
(2) the FD BC -> G ($X \rightarrow Z \cap R_2$ in Theorem 4.3) holds in R_1, or
(3) the EMVD C -> E|G ($X \cap R_2 \rightarrow\!\!\!\!\rightarrow Y \cap R_2 | Z \cap R_2$ in Theorem 4.3) holds in R_2, then R_1, R_2 preserves the EMVD BC ->-> DE|FG. Fig.4.7(a) shows the example relations corresponding to (1). Here, r_1 satisfies BC ->-> DE|FG and BC -> E, r_2 is an arbitrary relation on R_2, and the join $r_1 * r_2$ satisfies the EMVD BC ->-> DE|FG. Fig.4.7(b) also shows the example relations corresponding to (3). Here, r_1 satisfies BC ->-> DE|FG, r_2 satisfies C ->-> E|G, and $r_1 * r_2$ satisfies the EMVD BC ->-> DE|FG. Note that in this example, the EMVD C ->-> E|G holds in r_2, but does not hold in r_1. We assumed, however, that whenever a dependency holds in a relation scheme, it also holds in any other relation schemes on which the dependency can be defined. By this assumption, the conditions shown in Theorem 4.3 are simplified. Note that if the FD C -> E holds in R_2, then the EMVD BC ->-> DE|FG is preserved by $\{R_1, R_2\}$. But this assumption enforces that R_1 also satisfies the FD C -> E, and therefore R_1 becomes to satisfy the FD BC -> E (the condition shown in Theorem 4.3).

Theorem 4.4: Let $\{R_1, R_2\}$ be an arbitrary relational database scheme. Assume that the JD $*[S_1, \ldots, S_m]$ holds in R_1, where $R_1 = \bigcup_{i=1}^{m} S_i$. Then, R_1, R_2 preserves the EJD $*[S_1, \ldots, S_m]$ if the EJD $*[S_1 \cap R_2, \ldots, S_m \cap R_2]$ holds in R_2.
(Proof). Assume that r_1 is an arbitrary relation on R_1 that satisfy $*[S_1, \ldots, S_m]$, and that r_2 is an arbitrary relation on R_2 that satisfies $*[S_1 \cap R_2, \ldots, S_m \cap R_2]$. Suppose that the join $r_1 * r_2$ does not satisfy $*[S_1, \ldots, S_m]$. Then, $r_1 * r_2$ contains the tuples $t_1, \ldots, t_m$ (not necessarily distinct) and does not contain the tuple t such that

(a)

B	C	D	E	F	G
1	1	0	0	0	0
1	1	0	0	1	1
1	1	1	0	0	0
1	1	1	0	1	1

r_1

(b)

B	C	D	E	F	G
1	1	0	0	0	0
1	1	0	0	1	1
1	1	1	1	0	0
1	1	1	1	1	1
0	1	0	1	0	3
0	1	0	1	0	4

r_1

A	C	E	G
1	1	0	0
1	1	0	1
1	1	1	0

r_2

A	C	E	G
1	1	0	0
1	1	0	1
1	1	0	2
1	1	1	0
1	1	1	1
1	1	1	2

r_2

A	B	C	D	E	F	G
1	1	1	0	0	0	0
1	1	1	0	0	1	1
1	1	1	1	0	0	0
1	1	1	1	0	1	1

$r_1 * r_2$

A	B	C	D	E	F	G
1	1	1	0	0	0	0
1	1	1	0	0	1	1
1	1	1	1	0	0	0
1	1	1	1	0	1	1

$r_1 * r_2$

Fig.4.7. Examples for the preservation of $BC \twoheadrightarrow DE|FG$ by $\{R_1, R_2\}$.

$$t[S_1]=t_1[S_1],$$
$$t[S_2]=t_2[S_2],$$
$$\vdots$$
$$t[S_m]=t_m[S_m].$$

Since r_1*r_2 contains $t_1,\ldots,t_m$, r_1 also contains the tuples $t_1',\ldots,t_m'$ such that

$$t_1'[S_1]=t_1[S_1],$$
$$t_2'[S_2]=t_2[S_2],$$
$$\vdots$$
$$t_m'[S_m]=t_m[S_m].$$

r_1 satisfies the JD $*[S_1,\ldots,S_m]$, and therefore, r_1 also contains the tuple t' such that

$$t'[S_1]=t_1'[S_1],$$
$$t'[S_2]=t_2'[S_2],$$
$$\vdots$$
$$t'[S_m]=t_m'[S_m].$$

In the same way, r_2 is proved to contain the tuple t'' such that

$$t''[S_1 \cap R_2]=t_1[S_1 \cap R_2],$$
$$t''[S_2 \cap R_2]=t_2[S_2 \cap R_2],$$
$$\vdots$$
$$t''[S_m \cap R_2]=t_m[S_m \cap R_2].$$

Since r_1 and r_2 are joined on only $(\bigcup_{i=1}^{m} S_i) \cap R_2$, r_1*r_2 must contain the tuple t described above, which is equal to $t'*t''$. A contradiction. Q.E.D.

4.4 Dependency Preserving Normal Form

In this section, the notion of the dependency preserving normal form of a relation scheme is introduced. Roughly speaking, we say a relation scheme is in dependency preserving normal form if and only if some set of FDs can imply all the dependency constraints of the relation scheme. The name of the 'dependency preserving' normal form is due to the fact that any FD on a relation scheme is preserved as shown in Theorem 4.1. Given a relational database scheme $\{R_1,\ldots,R_n\}$ and a data dependency d, we show a necessary and sufficient condition for $\{R_1,\ldots,R_n\}$ to preserve d when every R_i is in dependency preserving normal form.

Definition 4.3: Let D_i be a given set of FDs, EMVDs and EJDs which hold in a relation scheme R_i. Let F_i be a set of FDs on R_i that are implied by D_i. A relation scheme R_i is said to be in Dependency Preserving Normal Form (for short, DPNF) if and only if F_i implies every dependency in D_i.

Theorem 4.5: Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme such that R_i is in DPNF for all i, $1\leq i\leq n$. Let D_i denote a set of FDs, EMVDs and EJDs which hold in R_i. Suppose that D is a given set of FDs, EMVDs and EJDs that can be defined on $R=\bigcup_{i=1}^{n} R_i$. Then, $\{R_1,\ldots,R_n\}$ preserves every dependency in D if and only if $\bigcup_{i=1}^{n} D_i \cup \{*[R_1,\ldots,R_n]\}$ implies D.
(Proof). Let F_i be a set of FDs on R_i that are implied by D_i for each i, $1\leq i\leq n$. From the assumption that each R_i is in DPNF,

$$SAT(F_i)=SAT(D_i)$$

for all i, $1 \leq i \leq n$. Let $*[R_i]$ denote the JD $*[R_1,\ldots,R_n]$ on R. Then,

$$\bigcup_{i=1}^{n} F_i \cup \{*[R_i]\} \models \bigcup_{i=1}^{n} D_i \cup \{*[R_i]\} \text{ and}$$
$$\bigcup_{i=1}^{n} D_i \cup \{*[R_i]\} \models \bigcup_{i=1}^{n} F_i \cup \{*[R_i]\}.$$

Let $G=\bigcup_{i=1}^{n} F_i \cup \{*[R_i]\}$. Then, it is sufficient to prove that $\{R_1,\ldots,R_n\}$ preserves D if and only if $G \models D$.

(if-part). For all i, $1 \leq i \leq n$, every FD in F_i is proved to be preserved by $\{R_1,\ldots,R_n\}$ form Theorem 4.1. Therefore, whenever r_i belongs to $SAT(F_i)$ for all i,

$$\overset{n}{*}_{i=1} r_i \in SAT(\bigcup_{i=1}^{n} F_i) \ldots\ldots(1)$$

holds. Furthermore, the join of r_i's, $\overset{n}{*}_{i=1} r_i$ satisfies the JD $*[R_i]$ since

$$\overset{n}{*}_{i=1} r_i = \overset{n}{*}_{j=1}((\overset{n}{*}_{i=1} r_i)[R_j]).$$

That is, whenever r_i belongs to $SAT(F_i)$

$$\overset{n}{*}_{i=1} r_i \in SAT(*[R_i]) \ldots\ldots(2).$$

Since $G=\bigcup_{i=1}^{n} F_i \cup \{*[R_i]\}$,

$$SAT(G)=SAT(\bigcup_{i=1}^{n} F_i) \cap SAT(*[R_i]) \ldots\ldots(3).$$

From (1), (2), (3) and the assumption that $G \models D$,

$$\overset{n}{*}_{i=1} r_i \in SAT(G) \subseteq SAT(D)$$

whenever r_i belongs to $SAT(F_i)$ for all i, $1 \leq i \leq n$. Therefore, $\{R_1,\ldots,R_n\}$ preserves every dependency in D.

(only-if-part). Assume that G does not imply D. That is,

$$SAT(G) \not\subseteq SAT(D).$$

There must exist a relation r in SAT(G)-SAT(D). Since r belongs to SAT(G), r satisfies the JD $*[R_i]$ and each $r[R_i]$ satisfies F_i. Therefore,

$$r = \overset{n}{\underset{i=1}{*}} r[R_i]$$

and, for all i, $1 \leq i \leq n$,

$$r[R_i] \in SAT(F_i).$$

From the assumption that $\{R_1,\ldots,R_n\}$ preserves D,

$$\overset{n}{\underset{i=1}{*}} r[R_i] \in SAT(D).$$

However, $\overset{n}{\underset{i=1}{*}} r[R_i] = r$ and therefore, r must belong to SAT(D). A contradiction. Q.E.D.

Example 4.2: Let R=ABCD be an initial relation scheme in which the following set D of data dependencies holds:

$$A \twoheadrightarrow D|BC,\ B \twoheadrightarrow C|AD.$$

The relation schemes R_1=AD, R_2=BC and R_3=AB can be obtained by Fagin's decomposition approach [FAGI7709]. That is, R_1 is obtained by the EMVD $A \twoheadrightarrow D|BC$, and R_2 and R_3 are obtained by

the EMVD B ->-> C|A (implied by B ->-> C|AD). Here, each F_i (i=1,2,3) is an empty set since D implies no nontrivial FDs. Furthermore, from Theorem 3.1, any nontrivial EMVD does not hold in any R_i. Therefore, R_1, R_2 and R_3 are in DPNF. Let us test whether or not the following holds:

$$\bigcup_{i=1}^{3} F_i \cup \{*[AD,BC,AB]\} \models D.$$

Recently, Mendelzon et al. [MENDM7910] showed that any JD is equivalent to a set of 'generalized mutual dependency' and some MVDs. By using their result, we can prove that the JD *[AD,BC,AB] is equivalent to {A ->-> D|BC, B ->-> C|AD}. Therefore, from Theorem 4.5, the relational databse scheme $\{R_1,R_2,R_3\}$ preserves D. Fig.4.8 (a) shows the decomposition process of R into R_1, R_2 and R_3.

Fig.4.8 (b) shows another decomposition of R, which produces the relation schemes R_1, R_2, R_3 and R_4=AC. These relation schemes are obtained by using the EMVDs AC ->-> B|D, B ->-> C|A and A ->-> C|D, which are implied by D. In this case, R_1, R_2, R_3 and R_4 are in DPNF. The JD *[AD,BC,AB,AC] can be proved to be equivalent to

$$\{A \text{ ->-> } D|BC,\ m[B,C,A]\}$$

by the result in [MENDM7910]. Here, m[B,C,A] is a mutual dependency that enforces for any relation r on R to be losslessly decomposable into r[BC], r[AB] and r[AC]. Obviously, the mutual dependency m[B,C,A] cannot imply the EMVD B ->-> C|AD, the condition in Theorem 4.5 does not hold. Therefore, the relational database scheme $\{R_1,R_2,R_3,R_4\}$ cannot preserve D.

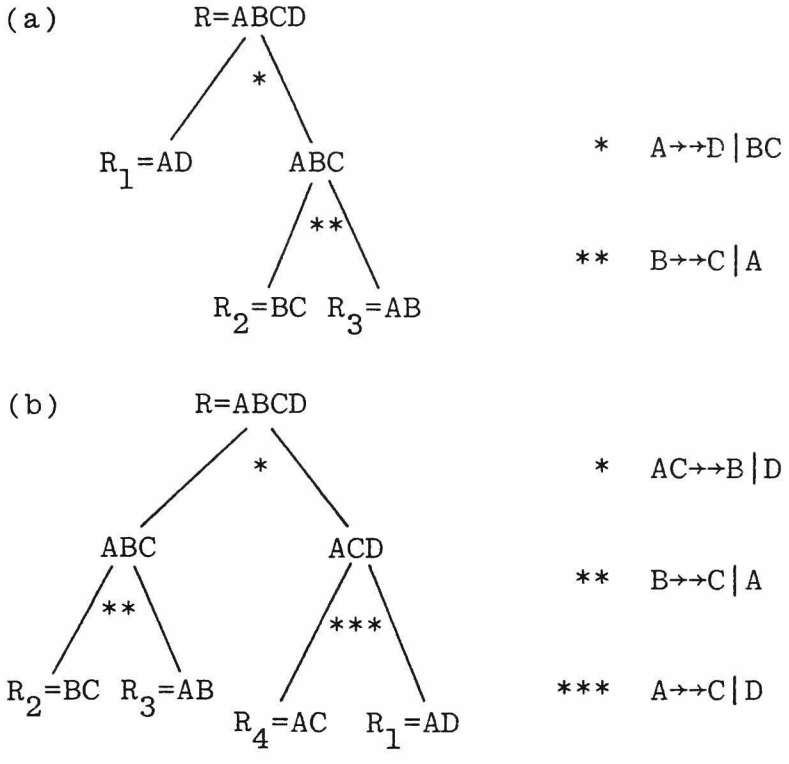


Fig.4.8. Decompositions of R=ABCD with D={A→→D, B→→C}.

The fact that a relation scheme is in DPNF does not imply that the relation scheme is even in BCNF. The reason why we introduce the DPNF is that if we drop out the assumption that every relation scheme is in DPNF, then the condition shown in Theorem 4.5 becomes only a necessary condition. The following example illustrates this problem.

Example 4.3: Let R=ABCD be an initial relation scheme such that D = {A ->-> D|BC, B ->-> C|AD}, as shown in Example 4.2. Assume that we have a relational database scheme {R_1=ABC,

R_2=ACD}, in which each relation scheme can be obtained by the EMVD AC ->-> B|D (implied by A ->-> D|BC). From the projection rule for EMVDs, the EMVD B ->-> C|A holds in R_1 and the EMVD A ->-> C|D holds in R_2. Therefore, neither R_1 nor R_2 is in DPNF. In this case, we can verify by the joinability rule for EMVDs that

$$D_1 \cup D_2 \cup \{*[ABC,ACD]\} \models D,$$

where D_1={B ->-> C|A} and D_2={A ->-> C|D}. From the EMVDs AC ->-> D|B (equal to the JD *[ABC,ACD]) and A ->-> C|D (in D_2), we can derive the EMVD A ->-> D|BC. By applying the EMVD2 rule to A ->-> D|BC, we can obtain AB ->-> C|D. From the EMVDs AB ->-> C|D and B ->-> C|A (in D_2), the EMVD B ->-> C|AD is derived. Fig.4.9 shows example relations r_1, r_2 and r_1*r_2, where r_1 satisfies B ->-> C|A, r_2 satisfies A ->-> C|D, but r_1*r_2 does not satisfy B ->-> C|AD in D. That is, $\{R_1,R_2\}$ cannot preserve D.

A	B	C
1	1	1
1	1	0
0	1	1
0	1	0

r_1

A	C	D
1	1	1
1	0	1
0	1	1

r_2

A	B	C	D
1	1	1	1
1	1	0	1
0	1	1	1

r_1*r_2

Fig.4.9. Example relations r_1, r_2 and r_1*r_2 such that $\{R_1,R_2\}$ does not preserve D={A ↠ D|BC, B ↠ C|AD}.

4.5 General Cases

In this section, we discuss the preservations of data dependencies for the following general case: A relational database scheme consists of not necessarily DPNF relation schemes.

The following theorem provides a necessary condition for a relational database scheme to preserve a data dependency. The proof is carried out in the same manner as the only-if-part of Theorem 4.5.

Theorem 4.6: Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme. Let D_i denote a set of FDs, EMVDs and EJDs which hold in R_i. If $\{R_1,\ldots,R_n\}$ preserves a data dependency d (FD, EMVD or EJD), then, $\bigcup_{i=1}^{n} D_i \cup \{*[R_1,\ldots,R_n]\}$ implies d.

(Proof). Let $*[R_i]$ denote the JD $*[R_1,\ldots,R_n]$. Assume that $G=\bigcup_{i=1}^{n} D_i \cup \{*[R_i]\}$ does not imply d. There must exist a relation r on $R=\bigcup_{i=1}^{n} R_i$ in SAT(G)-SAT(d) such that

$$r=\overset{*}{\underset{i=1}{\overset{n}{}}} r[R_i]$$

and for all i, $1\leq i\leq n$,

$$r[R_i] \in SAT(D_i).$$

From the assumption that $\{R_1,\ldots,R_n\}$ preserves d, if for all i, r_i belongs to $SAT(D_i)$, then

$$\overset{*}{\underset{i=1}{\overset{n}{}}} r_i \in SAT(d).$$

Therefore, $\overset{*}{\underset{i=1}{\overset{n}{}}} r[R_i]$ must also belong to SAT(d). That is, r must

belong to SAT(d). A contradiction. Q.E.D.

Furthermore, the following theorem provides a sufficient condition for a general relational database scheme to preserve a data dependency.

Theorem 4.7: Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme. Let F_i denote a set of FDs which hold in R_i for each i, $1\leq i\leq n$. Let d be a given data dependency which can be defined on $R=\overset{n}{\underset{i=1}{\cup}}R_i$. Then, if $\overset{n}{\underset{i=1}{\cup}}F_i \cup \{*[R_1,\ldots,R_n]\}$ implies d, then $\{R_1,\ldots,R_n\}$ preserves d (proof omitted).

The proof of Theorem 4.7 can be easily carried out in the same manner as the proof of the if-part of Theorem 4.5.

Now, let us consider what kind of FDs can be preserved by a relational database scheme. In Theorem 4.1, we prove that any FD, which holds in some relation scheme R_i, can be always preserved by any relational database scheme containing R_i. The remained problem is whether or not other new FDs can be produced by taking the join of the relations r_i's. The following theorem states that the answer is no, that is, any relational database scheme can preserve only the FDs that are implied by the data dependencies which hold in some relation schemes.

Theorem 4.8: Let $\{R_1,\ldots,R_n\}$ be an arbitrary relational database scheme. Let D_i denote a set of FDs, EMVDs and EJDs, that hold in R_i, for each i, $1\leq i\leq n$. Let f be an arbitrary FD statement which can be defined on $R=\overset{n}{\underset{i=1}{\cup}}R_i$. If $\{R_1,\ldots,R_n\}$ preserves f, then $\overset{n}{\underset{i=1}{\cup}}D_i$ implies f.

(Proof). From the results in [FAGI8003], there must exist a finite 'Armstrong relation' r on $R=\overset{n}{\underset{i=1}{\cup}}R_i$ for $\overset{n}{\underset{i=1}{\cup}}D_i$. That is, r

is a finite relation on R such that every dependency implied by $\overset{n}{\underset{i=1}{\cup}} D_i$ holds in r and any other data dependency (FD, EMVD, EJD) does not hold in r. Therefore, any FD, that holds in r, must be implied by $\overset{n}{\underset{i=1}{\cup}} D_i$. Let us consider the projections of r onto R_i's. For each i, $1 \leq i \leq n$, each projection $r[R_i]$ satisfies D_i, and therefore,

$$r[R_i] \in SAT(D_i).$$

We can prove that any FD, which holds in $\overset{n}{\underset{i=1}{*}} r[R_i]$, must be also implied by $\overset{n}{\underset{i=1}{\cup}} D_i$. Assume that some FD f holds in $\overset{n}{\underset{i=1}{*}} r[R_i]$, but that $\overset{n}{\underset{i=1}{\cup}} D_i$ does not imply f. Since $\overset{n}{\underset{i=1}{*}} r[R_i] \supseteq r$, f also holds in r. However, any FD, that holds in r, must be implied by $\overset{n}{\underset{i=1}{\cup}} D_i$. A contradiction. Q.E.D.

4.6 Concluding Remarks

In this chapter, we investigated what class of data dependencies can be preserved totally by a set of relation schemes. Several useful conditions for the preservability are shown under the following assumptions:

(a) For any relational database scheme, whenever a data dependency holds in a relation scheme, it also holds in any other relation scheme on which the dependency can be defined.

(b) For any relation scheme R_i, every relation r_i on R_i is allowed to be updated if and only if the resulting relation satisfies only the dependencies enforced on R_i.

(c) No inter-relational data dependency is given in advance for any relational database scheme. Here, inter-relational data dependencies mean dependencies that are defined among two or

more relation schemes.

The assumption (a) is concerned with so called 'universal relation scheme assumption'. It enforces that for any set of attributes, there must exist a unique set of data dependencies that hold in the set of attributes. The assumption (b) states that we can update each relation independently from others. In this sense, (a) and (b) are a relaxation of so called 'universal relation (or instance) assumption'. The assumption (c) may seem to be a strict restriction. We, however, have this assumption since such inter-relational data dependencies may often cause it impossible to update a relation independently from others.

The main results in this chapter are as follows:

(1) We showed conditions for MVDs or JDs holding in a relation scheme R to be preserved when we take a natural join of any relation r on R and any other relation.

(2) The notion of the Dependency Preserving Normal Form (DPNF) for a relation scheme is introduced. Under assumption that each relation scheme in a relational database scheme is in DPNF, we showed a necessary and sufficient condition that a data dependency is preserved by the relational database scheme.

As described in Section 4.3, several similar, but different concepts for preserving data dependencies have been introduced by many researchers [RISS7712], [BEERM7904], [MAIEM7912], [BEERR8001]. Especially, our results in Section 4.4 (Theorem 4.5) may seem to be similar to those in [MAIEM7912] or [BEERR8001]. Our results were, however, obtained independently from them and furthermore, their results are based on the 'universal relation assumption', but our results are not.The differences of our concept from others are summarized in Section 4.2.

It can be easily shown that any relation scheme in Codd's

BCNF or Fagin's 4NF is always in DPNF. Any relation scheme is possible to be decomposed into 4NF relation schemes, and thus, into DPNF relation schemes when only FDs and MVDs are given in advance. In this sense, the restriction that a relation scheme is in DPNF is not so strict. Generally, it will be, however, necessary to consider an algorithm to test whether or not a data dependency is preserved by a set of not necessarily DPNF relation schemes.

CHAPTER 5 SEMANTIC ASPECTS OF DATA DEPENDENCIES

In this chapter, we mainly discuss the semantic aspects of functional dependencies and multivalued dependencies in order to choose a better logical design of a relational database scheme [KAMBT7911S].

We clarify the differences between a conceptual design and a logical design of a relational database scheme. For example, a multivalued dependency does not always capture a conceptual dependency such that a data element semantically determines a set of other attribute values. Moreover, some transitively specified multivalued dependencies are shown to often impose a semantically 'unnatural' constraint. A sufficient condition for these multivalued dependencies to hold in a 'natural' sense is provided.

A mixed design approach of a conceptual design and a logical design is introduced, in which each design approach is compensated for its deficiency by the other one.

Some generalization of a relational model is also suggested to handle some interface problems between a logical design and a conceptual design, such as a null value problem and an entity identification problem.

5.1 Introduction

The overall conceptual description of a database is known as a conceptual schema [ANSI7502]. Several candidates to represent a conceptual schema have been introduced [CHEN7603], [SCHM7701] independently of the relational database design theory. On the other hand, Schmid and Swenson [SCHMS7505] and Hall et al. [HALL07601] modified the relational model so that

it can capture more semantics of data. Recently, Codd also introduced a data model [CODD7912], which is an extension of his relational model, in order to handle more semantics of data.

Apart from the discussions of the conceptual schema and semantics of data, much effort has been invested in the study of the relational database design theory [BEERB7809], [KAMB7906], [KAMBT7911S]. The semantic aspect of data dependencies have been, however, less studied.

In this chapter, we discuss the relationships between a conceptual schema and a logical design of a relational database scheme.

Schmid and Swenson pointed out the problems of functional dependencies (FDs) [SCHMS7505] and this leads to the introduction of multivalued dependencies (MVDs) by Fagin [FAGI7709] and independently by Zaniolo [ZANI7607]. The concept of FDs is successful to capture faithfully a class of functional mappings from a set of attribute values to a set of other attribute values. Except the null value problem of primary keys, there exists a one-to-one correspondence between FDs and the designer's conceptual dependency that a data element functionally determines a data element. We call the latter one the conceptual element-element dependency (for short, CEED).

Two approaches are considered to generalize the concept of FDs. One is the concept of MVDs, which is a necessary and sufficient condition of the information lossless decomposition of a relation into its two projections. It can be further generalized to the concept of join dependencies [AHO-B7909]. Note that the FD provides a sufficient condition for a relation to be decomposable losslessly into its two projections. Another approach is the concept of the conceptual element-set dependencies (for short, CESDs) [KAMBT7710], [KAMBT7801],

[KAMBT7805], [KAMBT7911S]. The CESD is a statement specified by database designers that each value of an attribute semantically determines a set of values of the other attribute. There does not always exist a one-to-one correspondence between a set of MVDs and a set of CESDs. These two approaches have the advantages and disadvantages as summarized in Fig.5.1.

	advantages	disadvantages
MVD	* theoretically complete * lossless decomposition * no ambiguity	* extenstion oriented * difficult to specify * null value problem
CESD	* easy to specify * empty set handling * intension oriented * basic information unit	* ambiguity

Fig.5.1. MVDs and CESDs.

The concept of MVDs has no ambiguity and is theoretically complete since it is defined as a necessary and sufficient condition of the lossless decomposition of a relation into its two projections. In this sense, it is a rather extension oriented concept and is suitable as integrity constraints. It leads to the introduction of the complementation rule [BEERF7708], which makes it difficult to specify correct MVDs [BISK78], [BEER7901]. Furthermore, as will be shown in this chapter, the transitively specified MVDs X->->Y and Y->->Z may often impose a relation to obey a semantically 'unnatural' constraint. The MVDs are also insufficient to handle null values since primary key attributes are not allowed to contain

null values and in this case, MVDs cannot represent a relationship between a value and an empty set of values.

The advantages and disadvantages of CESDs are, on the other hand, complementary to those of MVDs. Consider a relation scheme R={TEAM,PLAYER,CHILD} for which a database designer specifies that each team determines a set of its players, and that each player determines a set of his children. We denote these CESDs by TEAM=>=>PLAYER and PLAYER=>=>CHILD, respectively. In this sense, the CESD is a rather intension oriented concept and is suitable for representing basic information units [BEER7901]. As shown in Fig.5.2, the MVDs PLAYER->->TEAM and PLAYER->->CHILD hold in this relation. Note that in this relation, the MVD TEAM->->PLAYER does not hold, while the CESD TEAM=>=>PLAYER is specified. The CESDs can be always specified independently of the set of attributes constructing the relation scheme. Furthermore, the CESD allows a value associated with an empty set of values. For example, some player may be associated with an empty set of children since he has no children.

TEAM	PLAYER	CHILD
t_1	p_1	c_1
t_1	p_1	c_2
t_1	p_2	c_3
t_1	p_2	c_4
t_2	p_2	c_3
t_2	p_2	c_4
t_2	p_3	c_2

Fig.5.2. An example relation on R.

Although CESDs can be easily specified, they have some ambiguities even in those directions. In the example, other database designer may specify the CESDs PLAYER=>=>TEAM and PLAYER=>=>CHILD, which correspond to PLAYER->->TEAM and PLAYER->->CHILD, respectively.

Our results in this chapter are as follows:

(1) We show several problems of data dependencies in representing the semantics of data: Disagreements between conceptual and data dependencies, a set representation (null value) problem and an entity identification problem.

(2) The semantic analysis of FDs and MVDs is shown. Especially, transitively specified MVDs are shown to often impose a semanticall 'unnatural' constarint since they do not always correspond to the transitively specified CESDs. A sufficient condition for those MVDs to hold in a 'natural' sense is provided.

(3) We classify the relationships between a set of MVDs and a set of CESDs. A design approach with both conceptual and data dependencies is introduced to specify correct data dependencies and to choose a good logical design. The results are fed back to the design of conceptual dependencies.

(4) Some generalization of a relational model is suggested to handle the set representation problem. Especially, the null value problem of data dependencies are handled by the relaxation of primary key's condition. We also provide a sufficient condition for a relation with null values to be information-

5.2 Interface Problems

5.2.1 Disagreements between MVDs and CESDs

The important disagreements between MVDs and CESDs are caused by (a) the complementation rule for MVDs, (b) transitivity rule for MVDs and (c) the empty set representation problem (discussed in 5.2.3).

If a database designer wishes to translate a CESD X =>=> Y into the MVD X ->-> Y on the relation scheme R, then he should verify that that X ->-> R-XY also holds in R. Since this complementary MVD does not always hold, it causes a disagreement between MVDs and CESDs, and difficulties in specifying correct MVDs. In the relation scheme R={TEAM, PLAYER, CHILD, COACH, COACH'S-CHILD} in Fig.5.3, there does not exist an MVD corresponding to TEAM =>=> PLAYER or TEAM =>=> COACH. That is, neither TEAM ->-> PLAYER nor TEAM ->-> COACH holds. The MVDs that can represent these CESDs are

PLAYER ->-> CHILD
COACH ->-> COACH'S-CHILD
TEAM ->->{PLAYER,CHILD}.

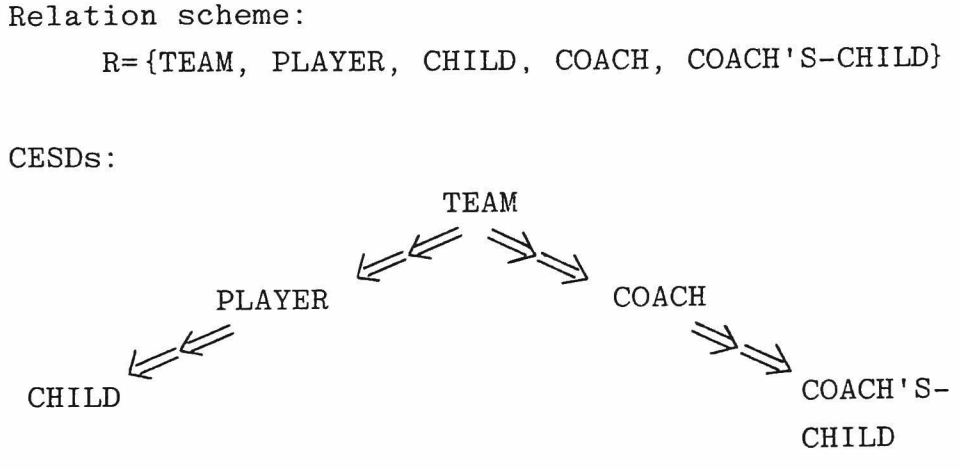


Fig.5.3. An example relation scheme with CESDs.

These MVDs can be shown to be equivalent to the set of the following MVDs and EMVDs:

TEAM ->-> PLAYER|COACH,
PLAYER ->-> CHILD|{TEAM,COACH},
COACH ->-> COACH'S-CHILD

by the results in [TANAK7908]. Note that the CESDs TEAM =>=> PLAYER and TEAM =>=>COACH correspond to the EMVD TEAM ->> PLAYER|COACH. Therefore, EMVDs are useful to find the MVDs equivalent to given CESDs. As for the transitivity rule, we have a quite different relation from Fig.5.2 if we specify both TEAM ->-> PLAYER and PLAYER ->-> CHILD (see Fig.5.4). This problem will be further discussed in Section 5.3.

5.2.2 Entity Identification Problem

This problem has been pointed out as a deficiency of the relational model [CHEN7603], [SCHM7701], [HALL07601]. An entity is a 'thing' which can be distinctly identified [CHEN7603]. Although the concept of keys is similar to the concept of the entity identifiers, the former cannot sufficiently represent the latter. This is because keys are used to uniquely identify not entities but tuples. Consider a relation scheme R={TEAM,PLAYER,CHILD} shown in Section 5.1, in which the MVDs PLAYER ->-> CHILD and PLAYER ->-> TEAM hold. In this case, can we allow the existence of more than one player (entity) with the same name? The answer is no if we regard those MVDs as correct. If we allow this, two distinct players with the same name may have distinct sets of childrens' names (see Fig.5.5), and then the MVD PLAYER ->-> CHILD does not hold in this scheme. On the

TEAM	PLAYER	CHILD
t_1	p_1	c_1
t_1	p_1	c_2
t_1	p_1	c_3
t_1	p_1	c_4
t_1	p_2	c_1
t_1	p_2	c_2
t_1	p_2	c_3
t_1	p_2	c_4
t_2	p_2	c_1
t_2	p_2	c_2
t_2	p_2	c_3
t_2	p_2	c_4
t_2	p_3	c_1
t_2	p_3	c_2
t_2	p_3	c_3
t_2	p_3	c_4

Fig.5.4. A relation on R= {TEAM,PLAYER,CHILD} in which

TEAM $\twoheadrightarrow$ PLAYER

and

PLAYER $\twoheadrightarrow$ CHILD

hold.

TEAM	PLAYER	CHILD
t_1	p_1	c_1
t_1	p_1	c_2
t_1	p_2	c_2
t_1	p_2	c_3
t_2	p_1	c_4
t_2	p_1	c_5

Fig.5.5. An example relation for the entity identification problem.

other hand, in this example, we can allow more then one child with the same name without violating those MVDs. See the CHILD-value c_2, which may mean that both players p_1 and p_2 have

distinct children with the same name.

One of our objectives is to construct a conceptual design easily mapped to the relational model. Hereafter, we have the following assumption concerned with the entity identification problem: If a CEED X => Y or a CESD X =>=> Y holds, then there should exist a one-to-one correspondence between a set of X-values and a set of entities.

5.2.3 Set Representation Problem

In normalizing a relation scheme, every attribute value is assumed to be a simple value. That is, each attribute value must not be a relation or a set. Conventional relational data languages, however, allow a kind of queries in which a simple value is regarded as a compound value. For example, if DATE-value '7305' means that the year is 1973 and that the month is May, then it is possible to handle DATE as two attributes YEAR and MONTH by numerical comparison operators. Furthermore, partial matching capabilities will make it possible to handle a string data as a compound value. On the other hand, there exists a case that it is difficult to translate a compound value into simple values. Especially when the number of components of a compound value is taken as an arbitrary finite one, it is not possible to translate easily the compound value into simple values (for example, consider a file name such as 'Yamada.Database.Minimum.File.2'). When the number of components is variable, it is possible to represent it by introducing null values. However, even if the maximum number of components is fixed, this attribute cannot be a primary key since a primary key is not allowed to contain null values [CODD7105F]. By these reasons, we can permit some kind of compound attribute as long

as

(1) such a compound value is a unit of updating, and

(2) there does not exist any data dependency from a proper subset of attributes constructing the compound attribute to other attributes.

Such a generalization is related to [MAKI7710].

A CESD X =>=> Y allows some X-value to be associated with an empty set of Y-values. Fig.5.6(a) shows an example in which A-value a_2 is associated with an empty set of B-values. This relation r can be decomposed into r[AB] and r[AC] (Fig.5.6(b) and (c)) without loss of information. Fagin's MVD is defined on 'all key' relation scheme when any FD does not hold, and null values are not allowed in such a scheme. Therefore, we relax the restriction of the existence of null values as follows: Each primary key value x of any tuple does not coincide with the one of any other tuple even if we replace the

(a)

A	B	C
a_1	b_1	c_1
a_1	b_2	c_1
a_1	b_1	c_2
a_1	b_2	c_2
a_2	-	c_2
a_2	-	c_3
a_3	b_2	-

(b)

A	B
a_1	b_1
a_1	b_2
a_2	-
a_3	b_2

(c)

A	C
a_1	c_1
a_1	c_2
a_2	c_2
a_2	c_3
a_3	-

Fig.5.6. Example relations with null values.

null values in x by possible values appearing in those attributes. Under the relaxed restriction, we provide a sufficient condition for a relation scheme with null values allowed to be losslessly decomposable as follows:

Theorem 5.1: [KAMBT7805]
Let r be any relation of R=XYZ, in which $r=r' \cup r''$ and X-value of r' contains no null value and any X-value of r'' is a concatenation of null values. r can be obtained by taking OR-join [ZANI7606] of r[XY] and r[XZ] if X ->-> Y|Z holds in r' and a nonexistence dependency from X to Y or from X to Z holds in r'' (here, the nonexistence dependency from X to Y means that if X-value is all null values, then Y-value must be also all null values).

5.3 Semantic Problems of Transitively Specified MVDs

In this section, two transitively specified MVDs are shown to often impose a semantically 'unnatural' constraint. A useful sufficient condition for the transitively specified MVDs to hold in a 'natural' sense is provided. We also provide some remarks about the decomposition of a relation scheme in which MVDs are transitively specified.

In [BEERF7708], Beeri et al. showed the transitivity rule for MVDs as follows: If both X ->-> Y and Y ->-> Z hold in R, then X ->-> Z-Y holds in R. This rule is valid in itself , and useful to derive a new MVD from two transitively specifiefd MVDs. It should be, however, noted that the transitively specified MVDs would impose a relation to obey the following constraint:

Theorem 5.2: Let X, Y and Z be nonempty disjoint sets of attributes. If X ->-> Y and Y ->-> Z hold in R, then r[x,Z]=r[x',Z] holds for each X-value x and x' such that

$$r[x,Y] \cap r[x',Y] \neq \phi.$$

(Proof). Suppose that R=XYZW and $XYZ \cap W = \phi$. Since the EMVDs X ->-> Y|ZW and Y ->-> Z|XW hold in R, from the projection rule for EMVDs [ZANI7606], [FAGI7709], the EMVDs X ->-> Y|Z and Y ->-> Z|X hold in R. That is, in any relation r of R,

$$r[x,Z]=r[xy,Z]$$

and

$r[y,Z]=r[yx,Z]$ hold in r for any XY-value xy. Assume that there exists some Y-value y in $r[x,Y] \cap r[x',Y]$ for some X-values x and x'. Then, we have

$$r[x,Z]=r[xy,Z],$$
$$r[x',Z]=r[x'y,Z],$$
$$r[y,Z]=r[yx,Z],$$
$$r[y,Z]=r[yx',Z].$$

We conclude that

$$r[x,Z]=r[x',Z]$$

holds in any relation r for these X-values x and x'. Q.E.D.

Theorem 5.2 states that if X ->-> Y and Y ->-> Z hold in R, and if there exists a Y-value associated with two X-values in some relation r of R, then the same set of Z-values should be associated with x and x' in r. Fig.5.7(a) shows an example relation in which X ->-> Y and Y ->-> Z hold. Fig.5.7(b) shows that the constraint imposed on mappings between X and Y, and between X and Z, by the two transitively specified MVDs.

On the other hand, two transitively specified CESDs X =>=> Y and Y =>=> Z do not cause this type of problem. This is because a class of CESDs is not semantically closed under the

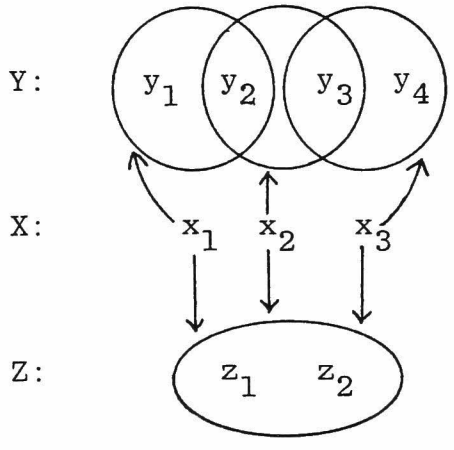
transitive connection. When two CESDs X =>=> Y and Y =>=> Z are specified, we can interpret that each X-value determines a set of sets of Z-values, not a set of Z-values.

In Theorem 5.2, we can replace X and Y by each other, and obtain the following corollary:

(a)

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_1
x_1	y_2	z_2
x_2	y_2	z_1
x_2	y_2	z_2
x_2	y_3	z_1
x_2	y_3	z_2
x_3	y_3	z_1
x_3	y_3	z_2
x_3	y_4	z_1
x_3	y_4	z_2

(b)



(c)

X	Y	Z
x_1	y_1	z_1
x_1	y_2	z_2
x_2	y_2	z_1
x_2	y_3	z_2
x_3	y_3	z_1
x_3	y_4	z_2

Fig.5.7.
The constraint imposed by $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$.

Corollary 5.1: If X ->-> Y and Y ->-> Z hold in R, then

$$r[y,Z]=r[y',Z]$$

holds for each Y-value y and y' such that

$$r[y,X] \cap r[y',X] \neq \phi.$$

The constraint derived in Theorem 5.2 is in itself a non-dependency constraint. Fig.5.7(c) shows an example relation in which the constraint does not imply any non-trivial MVD by itself. In the following theorem, we show that one of the transitively specified EMVDs is equivalent to the non-dependency constraint under the existence of the other EMVD.

Theorem 5.3: Assume that X ->-> Y|Z holds in R=XYZW, where $XYZ \cap W = \phi$ and X,Y,Z are nonempty disjoint sets. Then, the following constraints are equivalent:

(1) Y ->-> Z|X holds in R.

(2) r[x,Z]=r[x',Z] holds for each X-value x and x' such that $r[x,Y] \cap r[x',Y] \neq \phi$.

(Proof). It is obvious to prove that (1) implies (2) (see Theorem 5.2). If there do not exist any distinct X-values x and x' in any r such that $r[x,Y] \cap r[x',Y] \neq \phi$, then the FD Y -> X should hold in R. Since the FD Y -> X implies the EMVD Y ->-> X|Z to hold in R. Otherwise, assume that Y ->-> Z|X does not hold in R. In some relation r, there must exist subtuples (x,y,z) and (x',y,z'), but at least one of the subtuples (x',y,z) and (x,y,z') does not exist. Assume that the subtuple (x,y,z') does not exist. In r,

$$r[x,Y] \cap r[x',Y] \neq \phi$$

since y is contained in r[x,Y] and r[x',Y]. From the assumption (2),

r[x,Z]=r[x',Z]

should hold in r, and therefore, r must contain the subtuple (x,y,z'). A contradiction. Q.E.D.

Recently, Nicolas has shown an example of an interaction between an FD and a non-dependency constraint [NICO7805]: If X -> Y holds and if s is a symmetric binary relation of S=XY, then the FD Y -> X holds in s. There exists some analogy between Theorem 5.2 and this example. The difference is that in this example, the non-dependency constraint that s be symmetric is not equivalent to Y -> X even if X -> Y holds.

The constraint shown in Theorem 5.2 is a semantically 'unnatural' constraint because neither X-value nor Y-value can determine a set of Z-values independently from each other. Some specific FDs can, however, remedy this semantic problem. The FDs X -> Y and Y -> X are restated as in any r
(a) $r[y,X] \cap r[y',X] = \phi$ for any distinct Y-values y and y', and
(b) $r[x,Y] \cap r[x',Y] = \phi$ for any distinct X-values x and x', respectively. Therefore, we have the following theorem:

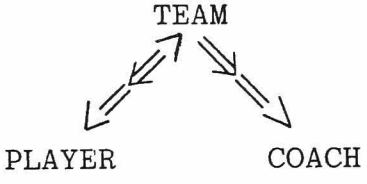
Theorem 5.4: Let X, Y and Z be nonempty disjoint sets. Assume that X ->-> Y and Y ->-> Z hold in R. If X -> Y or Y -> X holds in R, then the semantic problem of the transitively specified MVDs does not occur (proof omitted).

Let us assume that the relation scheme R={TEAM, PLAYER, COACH} obeys the following assumptions: Each team determines a set of its players uniquely, and each player can belong to only one team. Each team also determines a set of its coaches uniquely, and each coach can belong to one or more teams. In this scheme, TEAM ->-> PLAYER and PLAYER ->-> COACH (implied by

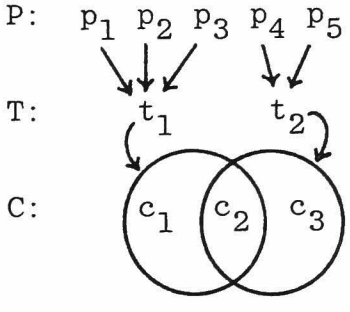
the FD PLAYER -> TEAM) hold. However, the semantic problem of the transitively specified MVDs does not occur since PLATER -> TEAM also holds in R. Fig.5.8(a) shows the corresponding CEEDs and CESDs. Fig.5.8(b) shows an example mapping and Fig.5.8(c) shows an example relation of R.

Theorem 5.4 provides a sufficient condition for two transitively specified MVDs to hold in a 'natural' sense. Next, we show some remarks in decomposing a relation scheme in which two transitively specified MVDs hold. For simplicity, we consider a relation scheme R=XYZ, where X, Y and Z are nonempty disjoint sets and X ->-> Y and Y ->-> Z hold in R.

(a) CEED & CESDs.



(b) data mappings.



(c) An example relation.

T	P	C
t_1	p_1	c_1
t_1	p_1	c_2
t_1	p_2	c_1
t_1	p_2	c_2
t_1	p_3	c_1
t_1	p_3	c_2
t_2	p_4	c_2
t_2	p_4	c_3
t_2	p_5	c_2
t_2	p_5	c_3

Fig.5.8. An example in which TEAM ↠ PLAYER and PLAYER ↠ COACH hold in a natural sense.

Remark 5.1: If neither X -> Y nor Y -> X holds in R, then any decomposition of R causes a serious problem when update operations are performed. Two lossless decompositions of R are possible in this case: One is a decomposition by X ->-> Y|Z to produce P=XY and Q=XZ. The other lossless decomposition is by Y ->-> Z|X to produce P=XY and S=YZ. The latter design seems to be a better design since X ->-> Y and Y ->-> Z are explicitly embodied by P=XY and S=YZ, respectively. As shown in Theorem 5.2, however, the implied constraint (r[x,Z]=r[x',Z] must hold for each X-value x and x' such that r[x,Y] $\cap$ r[x',Y]$\neq\phi$) cannot be maintained independently by P=XY or S=YZ when update operations are performed to a relation on P or on S. By the similar reason, the former design is not also sufficient.

Remark 5.2: Assume that Y -> X holds in R=XYZ. Although the semantic problem does not occur in this case, we should be careful to choose a data dependency to decompose the relation scheme R. One lossless decomposition is by Y -> X to obtain P=XY and S=YZ. The other is by X ->-> Y|Z to obtain P=XY and Q=XZ. The former one does not explicitly represent X ->-> Y|Z since X ->-> Y|Z is not necessarily hold in the join of relations p on P and s on S after update operations are performed to p or s. The latter one is a good design since P=XY explicitly represents Y -> X and furthermore, the join of relations p on P and q on Q always guarantee X ->-> Y|Z even after any update operations. Therefore, it should be noted that in this case, the EMVD X ->-> Y|Z has the priority over the FD Y -> X in their application to the decomposition of R.

5.4 Interactions between Conceptual and Data Dependencies

In this section, we show the usage of conceptual dependencies and the semantic analysis of data dependencies in a relational database scheme design.

Fig.5.9 shows our approach to designing a relational database scheme. At the level of conceptual dependency design, a database designer specifies CEEDs and CESDs by reference to several conceptual models such as Chen's Entity-Relationship model [CHEN7603]. From the CEEDs and CESDs, data dependencies (FDs and MVDs) are designed at the next level. At the succeeding level, the specified data dependencies are analyzed and checked for their semantic correctness. By this analysis, it is possible (a) to find more data dependencies necesary for the given data dependencies to hold in a ntural sense (for example, see Section 5.3), (b) to find semantically illegal data dependencies (for example, some MVDs must be, in fact, FDs and some CESDs must be CEEDs etc.) and (c) to remove redundantly specified data dependencies. These results are fed back to the design of conceptual and data dependencies.

In designing a set M of MVDs from a set C of given CESDs, we can classify the relationships between M and C as follows:

(Case A) Each CESD X =>=> Y corresponds to an MVD X ->-> Y, and each MVD X ->-> Y corresponds to a CESD X =>=> Y.

(Case B) Case A does not hold, but M faithfully represents C in the relation scheme consisting of all the attributes appearing in M.

(Case B-1) Each CESD X =>=> Y is explicitly represented by a relation scheme R=XY, which is obtained by the decomposition using M.

(Case B-2) Case B-1 does not hold.

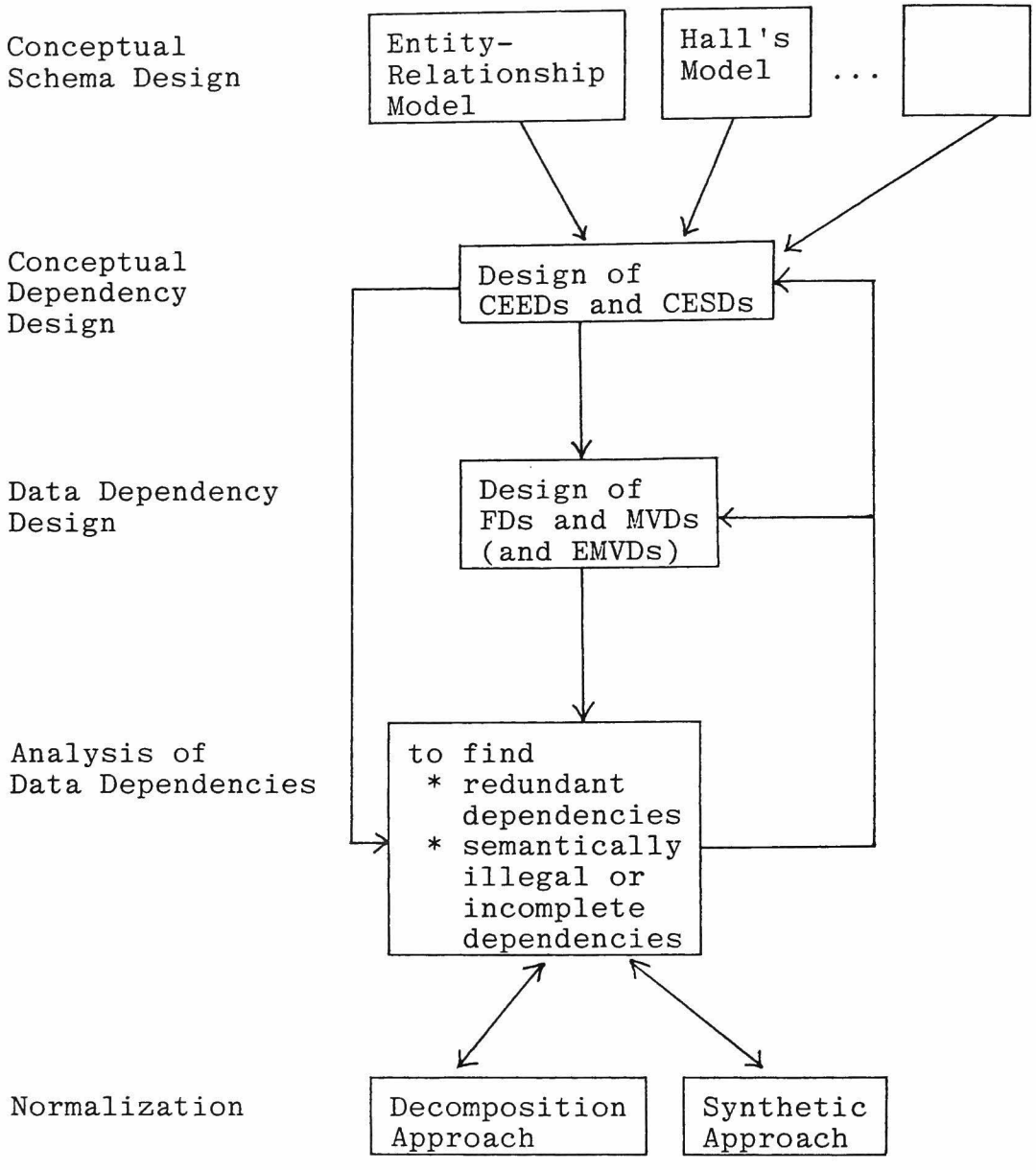


Fig.5.9 A relational database design with conceptual and data dependencies.

(Case C) There does not exist a set of CESDs that can represent a given set M of MVDs.

(Case D) There does not exist a set of MVDs that faithfully represents a given set C of CESDs.

The example shown in Fig.5.3 corresponds to Case B-1. By using the MVDs shown in this example, we can obtain the relation schemes:

R_1={PLAYER,CHILD}

R_2={TEAM,PLAYER}

R_3={TEAM,COACH}

R_4={COACH,COACH'S-CHILD}.

Each CESD shown in Fig.5.3 is explicitly represented by one of the relation schemes listed above.

Fig.5.10 illustrates an example design by our approach. This example is cited from [CHEN7603], and corresponds to Case B-2. The CEEDs and CESDs are

EMP# ==> NAME,

PROJ# ==> DESC (description),

PROJ# =>=> MGR#,

EMP# =>=> PROJ#,

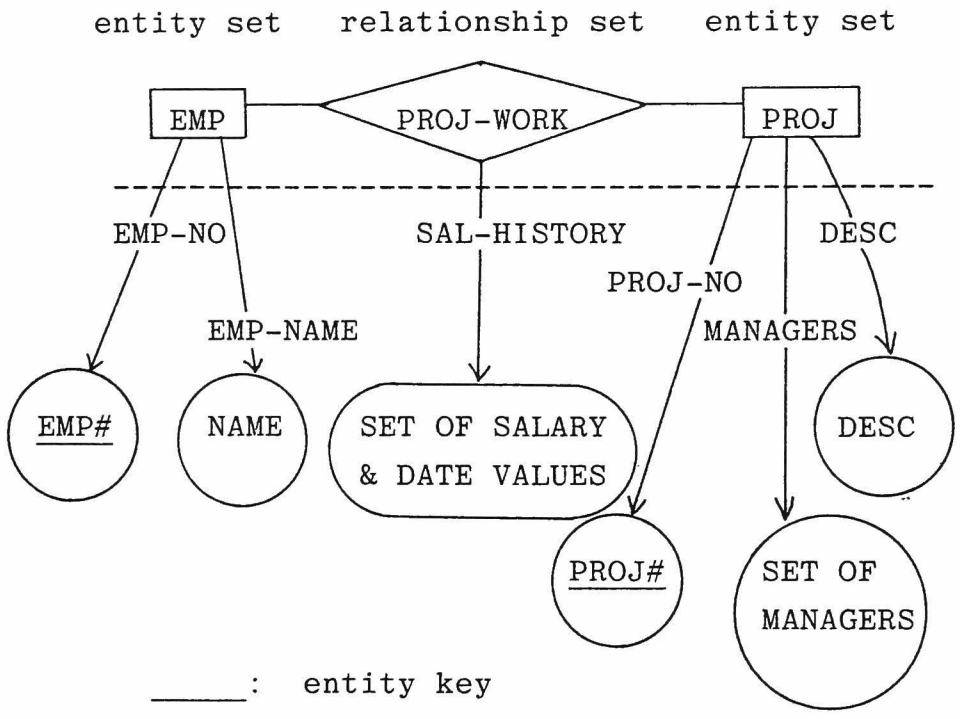
{EMP#,PROJ#} =>=> {SAL,DATE}.

Initially, assume that these CEEDs and CESDs are directly translated as FDs and MVDs, respectively. The analysis of data dependencies results in

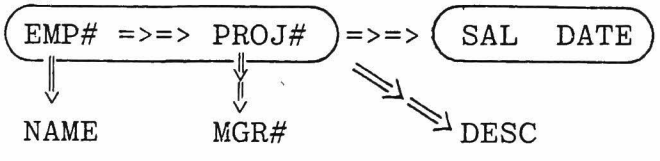
(1) EMP# ->-> PROJ# may be semantically illegal since it causes the semantic problem of transitively specified MVDs together with PROJ# ->-> MGR#.

(2) {EMP#,PROJ#} ->-> {SAL,DATE} is regarded as a redundant MVD from the following derivation by the inference rules for FDs and MVDs [BEERF7708]:

Chen's Entity-Relationship Diagram



CEEDs and CESDs



FDs and MVDs

EMP# → NAME
PROJ# → DESC
PROJ# ↠ MGR#
{EMP#,PROJ#} ↠ {SAL,DATE} : redundant
EMP# ↠ PROJ# : semantically illegal

Fig.5.10 An example design by our approach.

PROJ# ->-> MGR#

↓(MVD0 rule)

PROJ# ->-> {EMP#,NAME,SAL,DATE,DESC}

↓(MVD2 rule)

{PROJ#,EMP#} ->-> {NAME, SAL, DATE, DESC}...(a)

EMP# -> NAME, PROJ# -> DESC

↓(FD-union and FD-augmentation rules [BEERF7708])

{PROJ#,EMP#} -> {NAME,DESC}

↓(FD-MVD rule [BEERF7708])

{PROJ#,EMP#} ->-> {NAME,DESC}...(b)

From (a) and (b),

{PROJ#,EMP#} ->-> {SAL,DATE}.

(3) The CESD EMP# =>=> PROJ# is not explicitly represented by a relation scheme unless the trivial MVD {EMP#,PROJ#} ->-> ϕ is used in the decomposition of R={EMP#, PROJ#, NAME, MGR#, SAL, DATE, DESC}. In this case, by only nontrivial FDs and MVDs, we obtain

R_1={EMP#, NAME},

R_2={PROJ#, DESC},

R_3={PROJ#, MGR#},

R_4={EMP#, PROJ#, SAL, DATE}.

Case C corresponds to the case in which two MVDs X ->-> Y and Y ->-> Z are transitively specified. If we regard these MVDs as correct ones, and if neither X -> Y nor Y -> X holds, we cannot represent them by CESDs.

As an example of Case D, let the CESDs X =>=> Y, X =>=> Z

and XYZ =>=> W be given. In this case, there do not generally exist MVDs that represent these CESDs. The concept of EMVDs are, however, useful to handle such a case. The CESDs of the example are represented by the following EMVDs:

X ->-> Y|Z,

XYZ ->-> W|$\not\emptyset$.

5.5 Concluding Remarks

In this chapter, we discuss semantic aspects of data dependencies in designing a relational database scheme. A design approach with both conceptual and data dependencies is introduced to specify correct data dependencies, and to choose a good logical design. Some generalization is also suggested to handle more semantics of data by data dependencies. Further classification of conceptual dependencies is provided in [KAMBT7710].

Recently, Mendelzon has shown a minimal complete set of inference rules for MVDs: complementation, reflexivity and transitivity rules [MEND7901]. On the semantic aspects of MVDs, we have shown that the transitivity rule often causes a serious problem as well as the complementation rule. Further research will be needed on the relationships between the semantic aspects and the syntactic aspects (inference rules) of MVDs.

As for the problem of transitively specified MVDs, recently, Katsuno [KATS8003] and Nakamura, Chen [NAKAC8011] provided further investigations on our results in Section 5.3. Katsuno showed that the problem can be solved by considering union operations in some cases. Nakamura and Chen generalized our Theorems 5.2 and 5.3, and discussed the relationships between the transitivity problem and decompositions of BCNF relation schemes.

CHAPTER 6 ORGANIZATION OF QUASI-CONSECUTIVE RETRIEVAL FILES

In 1972, Ghosh introduced the consecutive retrieval (CR) file organization. It is an efficient file organization in which all records pertinent to each query are consecutively stored on linear storage locations. In this chapter, we introduce the quasi-consecutive retrieval (QCR) file organization. The QCR file organization is an extension of Ghosh's CR file organization and normally offers less redundancy. In the QCR file, all the records pertinent to each query are not necessarily stored consecutively, but rather they are stored within an area of a given buffer size. We mainly discuss graph theoretic properties of CR files and QCR files by investigating the properties of interval graphs well known in graph theory. We provide a basic condition for the existence of a QCR file without redundancy for a given buffer size. By introducing the notion of the redundant queries, such a condition is simplified. Furthermore, a heuristic computer algorithm is given to organize a QCR file with less redundancy for a given buffer size.

6.1 Introduction

In database systems, both the amount of required storage space and the retrieval time are important criteria of performance. They depend very much on how data is physically organized as a file on a computer. Since there generally exists a trade-off between these two criteria, several file organization techniques have been introduced, each with a particular goal, that is, either to reduce the redundancy of a

file or to reduce the number of accesses to auxiliary storage devices.

In this chapter, we introduce a new file organization by which both the retrieval time and the amount of storage space will be effectively reduced. Such a file organization is called the quasi-consecutive retrieval file organization (for short, the QCR file organization) [TANAK7703] [TANAK79].

In 1972, Ghosh introduced the consecutive retrieval file organization (for short, the CR file organization) [GHOS7209]. This is an efficient file organization in which all the records pertinent to each query are consecutively stored on linear storage locations. If it is possible to organize a CR file without any record duplication, it provides the minimum overall retrieval time for all queries using the minimum storage space. Unfortunately, it is, however, generally necessary to use redundant storage space for duplicates of records in order to organize a CR file. Hence, Ghosh suggested that a CR file be organized by partitioning the set of queries into a number of subsets such that a CR file is organized for each of these subsets. The least number of these subsets and an organization method are given by Yamamoto et al. [YAMAU77]. Waksman and Green [WAKSG7402] considered a file organization by which a set of records pertinent to each query could be retrieved within specified number of accesses, with the condition that in each access, pertinent records are consecutively retrieved.

An inverted file organization can be said to be a CR file with redundancy. Though it possesses greater flexibility and achieves fast retrieval, it does require a great amount of redundant storage space.

For example, assume that there are seven records r_1, r_2, r_3, r_4, r_5, r_6, r_7 and nine queries q_1, q_2, q_3, q_4, q_5, q_6, q_7,

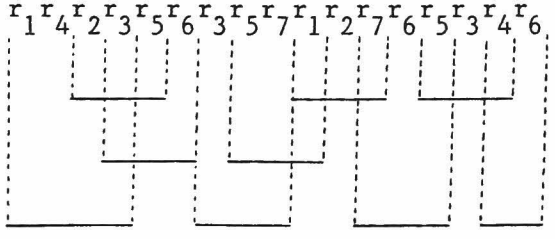
q_8, q_9 such that

$$R(q_1)=\{r_1,r_2,r_3,r_4\},$$
$$R(q_2)=\{r_2,r_3,r_5\},$$
$$R(q_3)=\{r_3,r_5,r_6\},$$
$$R(q_4)=\{r_3,r_5,r_7\},$$
$$R(q_5)=\{r_1,r_2,r_7\},$$
$$R(q_6)=\{r_1,r_5,r_7\},$$
$$R(q_7)=\{r_5,r_6,r_7\},$$
$$R(q_8)=\{r_3,r_4,r_5\},$$
$$R(q_9)=\{r_4,r_6\}.$$

Here $R(q_j)$ denotes a set of records in which each record is pertinent to a query q_j. For this example, it is possible to construct an inverted file by placing the records in each $R(q_j)$ consecutively as follows:

$$r_1r_2r_3r_4/r_2r_3r_5/r_3r_5r_6/r_3r_5r_7/r_1r_2r_7/r_1r_5r_7/r_5r_6r_7/r_3r_4r_5/r_4r_6,$$

where '/' is used as a delimiter to separate each $R(q_j)$. Let the redundancy of a file F be $|F|-|R|$, where $|F|$ and $|R|$ are the number of records in F and the number of distinct records in F, respectively. The redundancy of the inverted file obtained above is 20. By overlapping the records in $R(q_i) \cap R(q_j)$ $(i \neq j)$ appropriately, we can obtain a CR file with less redundancy as follows:



where each underline shows a set of records pertinent to each query. In this case, all the records in $R(q_1)$ are placed from the 1st to the 4th position, and all the records in $R(q_2)$ are placed from the 3rd to the 5th position. That is, the records in $R(q_1) \cap R(q_2)=\{r_2,r_3\}$ are overlapped. Note that the redundancy of this file is reduced to 10. For this example, it is impossible to construct a CR file with the redundancy 0 (without redundancy), since there exists no permutation of the seven records in which for each j, all the records in $R(q_j)$ are placed in the consecutive positions [GHOS7209].

Our QCR file organization is one of the generalizations of the CR file organization. In the QCR file organization, all the records pertinent to each query are not necessarily stored consecutively. Instead, they are stored within an area of the specified length on linear storage locations. Since data is usually transferred from auxiliary storage devices, such as disks and tapes to the user's working area by way of a buffer area, each set of records pertinent to a query is stored within an area whose size is equal to that of the buffer area. This generalization is based on the concept of the buffer area and is due to the assumption that the size of data obtained by one access is determined by the size of the buffer area.

We can reduce the redundancy further if we organize a QCR file for the example above. Let us assume that the buffer area can contain at most four records. Then, for example, we can obtain the QCR file with three duplicate records as follows:

$$r_6r_1r_4r_2r_3r_5r_6r_7r_1r_2.$$

In this QCR file, we can find a sequence of records of length four or less, which contains all the records in $R(q_j)$ for any

q_j.

For any given set of queries and records, it is always possible to organize the QCR file without any redundancy if we set the buffer size at the number of distinct records. Since such a QCR file is not practical, we consider the following problem:

(1) determine whether or not it is possible to organize a QCR file without redundancy for a specified buffer size and how to organize it if possible.

In a QCR file, after transferring data including pertinent records to the user's working area, it is necessary to check the data and to extract only pertinent records. Since the buffer size is related to the number of records to be checked, it is also necessary to consider the following problem:

(2) organize a QCR file without redundancy by the minimum buffer size.

By appropriate duplicates of records, it is always possible to organize a QCR file for a specified buffer size. In this case, it will be necessary to consider the following problem:

(3) organize a QCR file with the minimum redundancy for a specified buffer size.

Eswaran was among the first to develop a graph theoretical approach for analyzing the CR property [ESWA7503] [GHOS76]. In this chapter, we discuss the relationship between CR files and interval graphs which are well known in graph theory. The problems (1) and (2) are especially related to such graphs. In our approach, by introducing redundant queries, each of which identifies each record, several properties concerned with interval graphs and the CR files are simplified. The properties of the QCR file are explained from the standpoint of graph theory, and as a result, we deduce a basic property for

organizing a QCR file without redundancy for a given buffer size. Though there exists a linear algorithm [BOOTL7505] [BOOTL7612] for testing and organizing a CR file, it is, however, suspected that none of the problems (1), (2) and (3) can be solved in less than exponential time.

The relationship between a set of records and a set of queries is represented by a record-query (0,1) incidence matrix M defined in the following section. A CR file corresponds to a permutation of rows of M such that all 1's are consecutively placed in each column. The problem of organizing a QCR file for a given buffer size can be transformed into a problem of organizing a CR file after rewriting appropriate 0 elements in M into 1 elememnts. A candidate set of 0 elements to be rewritten can be combinationally selected and it seems to require exponential time in order to choose the set of 0 elements to be rewritten, to satisfy a restriction on a buffer size.

In this chapter, we assume that both the set of queries and the set of records are arbitrary. That is, no restriction is imposed on the structure of the record-query incidence matrix. The properties of query structure, such as Ghosh's 'combinatorial query set of order m'[GHOS7508], is beyond the scope of this paper.

A large amount of data must be organized as files in practical database systems. If it can be organized in a reasonable time, a QCR file, in which the number of duplicates of records is minimized, can be considered an important and useful contribution. We have therefore developed a computer algorithm to organize a QCR file with less redundancy. We took a heuristic approach by which the problem is divided into two subproblems: a partition problem for a set of queries and a placement problem for a set of records. It is also possible to

use the solution for each subproblem directly as a QCR file.

The partition problem is solved by the techniques similar to those in the partition of logic circuits in design automation etc. Hence, several algorithms are available for this problem. A set Q of queries is partitioned into $Q_1,\ldots,Q_p$ such that for each j (j=1,2,...,p), the total number of records pertinent to at least one query in Q_j is not more than k (the buffer size). In our algorithm, the number p is approximately minimized.

Next, the placement problem is to find a sequence of records, in which every set of records pertinent to each subset of queries obtained above are consecutively stored and the number of overlapped records is approximately maximized. For this problem, two algorithms are provided: one is to decide the initial arrangement of records, the second is to increase the number of overlapped records for this initial arrangement. Since the initial arrangement is also designed to make the number of overlapped records large, each solution of the algorithms introduced here can be used directly as a QCR file.

6.2 Characteristics of the QCR File

In this section, basic definitions and the characteristics of the QCR file are provided together with examples.

Definition 6.1: A finite set of distinct records and a finite set of distinct queries are denoted by

$$R=\{r_1,r_2,\ldots,r_m\}$$

and

$$Q=\{q_1,q_2,\ldots,q_n\},$$

respectively. A set of records pertinent to query q_j is denoted by $R(q_j)$. A set of queries each of which requires the record r_i

to be a pertinent record is denoted by $Q(r_i)$. We assume that there exists no q_j such that $R(q_j)$ is empty and no r_i such that $Q(r_i)$ is empty.

Definition 6.2: For given R and Q, a record-query incidence matrix M (for short, RQ matrix) is defined as follows; M is an m by n matrix whose rows correspond to the m records in R and whose columns correspond to the n queries in Q. The (i,j)th element of M contains 1 if r_i is pertinent to q_j, and 0 if not.

Suppose that a given RQ matrix is in a form as shown in Fig.6.1. In this figure, we can find that for example, $R(q_1)$ is $\{r_1, r_2, r_3, r_4\}$ and inversely, $Q(r_1)$ is $\{q_1, q_5, q_6\}$.

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
r_1	1	0	0	0	1	1	0	0	0
r_2	1	1	0	0	1	0	0	0	0
r_3	1	1	1	1	0	0	0	1	0
r_4	1	0	0	0	0	0	0	1	1
r_5	0	1	1	1	0	1	1	1	0
r_6	0	0	1	0	0	0	1	0	1
r_7	0	0	0	1	1	1	1	0	0

M

Fig.6.1. An example of the RQ matrix.

It is assumed that each record in R has the same size and that a buffer can contain at most k records. 'k' is called a buffer size. We generalize the consecutive retrieval file introduced by Ghosh [GHOS7209] using the concept of the buffer size and define the quasi-consecutive retrieval file.

Definition 6.3: A Consecutive Retrieval file (for short, a CR file) is defined as a sequence of records in which for each query q_j, there exists a subsequence such that all the records in $R(q_j)$ are consecutively placed. A Quasi-Consecutive Retrieval file (for short, a QCR file) is defined as a sequence of records in which there exists a subsequence of records for each query q_j such that all the records in $R(q_j)$ is contained and its length is less than or equal to a given buffer size. A QCR file without redundancy means a QCR file which consists of distinct records. It corresponds to the permutation of rows of M such that the length of the area in every column between the uppermost 1 and the lowermost 1 (this area is called an retrieval area) is less than or equal to k.

Definition 6.4: If there exists a permutation of rows for a given (0,1) matrix, by which the ones in every column are placed in consecutive positions after rewriting appropriate h zeros into ones, such a matrix is said to have h-CR property. Especially when h=0, such a matrix is said to have a consecutive retrieval (CR) property.

As for the h-CR property of M, obviously, we have the following property. Let us assume that for a given M, the number of zeros within each retrieval area is denoted by z_j, $0 \leq z_j \leq m-2$. If the summation of the z_j's is less than or equal to

h for a given M, then M has a h-CR property. If M has a h-CR property, then M also has a h'-CR property such that

$$h \leq h' \leq m * n - \sum_{j=1}^{n} |R(q_j)|.$$

If M does not have a h-CR property, then M does not have a h''-CR property such that $h'' \leq h$, either.

For the example M in Fig.6.1, we can find that M has a 13-CR property since

$$\sum_{j=1}^{9} z_j = 13.$$

6.3 Graph Theoretical Properties on CR Files and QCR Files

Since both CR files and QCR files are closely related to interval graphs in graph theory, their properties are discussed and clarified by a graph theoretical approach in this section.

Eswaran developed a graph theoretical approach for analyzing the CR property and established some necessary conditions for the existence of the CR property [ESWA7503] [GHOS76]. Our main objective is to clarify the relationship between the CR file and the QCR file - both of which have no redundancy. For this objective, in this section, the necessary and sufficient condition for the existence of the CR property of a given RQ matrix is provided from the standpoint of graph theory. Introducing the redundant queries in order to correspond the CR file with an interval graph, such a condition is simplified. The relationship between the CR file and the QCR file is also clarified by the redundant queries.

6.3.1 Definitions

Assume that an undirected graph treated in this chapter has

neither a self-loop nor multiple edges. An undirected edge connecting vertices a,b is represented as (a,b). A directed edge from a to b is represented as [a,b].

Definition 6.5: A cycle [GILMH64] of un undirected graph G is defined as any finite sequence of vertices $v_1,v_2,\ldots,v_k$ of G such that all of edges (v_i,v_{i+1}), $1\leq i\leq k-1$, and the edge (v_k,v_1) are in G, and for no vertices a and b and integers $i,j<k$ $(i\neq j)$, $a=v_i=v_j$, $b=v_{i+1}=v_{j+1}$ or $a=v_i=v_k$, $b=v_{i+1}=v_1$. A cycle $v_1,v_2,\ldots,v_k$ is called an odd cycle or even cycle depending on whether k is odd or even. If all the vertices in a cycle are distinct, such a cycle is, in particular, called a simple cycle. By a triangular chord [GILMH64] of a cycle $v_1,v_2,\ldots,v_k$ of G is meant any one of edges (v_i,v_{i+2}), $1\leq i\leq k-2$, or (v_{k-1},v_1) or (v_k,v_2).

The definition of a cycle is a little different from usual ones. That is, there can exist cycles in which a vertex appears more than one time, but any same ordered pair of vertices cannot appear in a cycle.

Definition 6.6: Let $<$ be a non-reflexive partial ordering defined on a set P. Let G(P,<) be the undirected graph whose vertices are the elements of P, and whose edges connect a and b for which either $a<b$ or $b<a$. A graph G with vertices P for which there exist a partial ordering $<$ such that $G=G(P,<)$ is called a comparability graph [GILMH64].

Definition 6.7: An undirected graph G is an interval graph [BOOTL7612] if and only if there is a one to one correspondence between its vertices and a set of intervals on the real line such that two vertices are adjacent if and only if the

corresponding intervals have a nonempty intersection.

Definition 6.8: A clique is a maximal complete subgraph in a given graph. For a given undirected graph G, its clique versus vertex incidence matrix is defined as follows. It is an m by n matrix whose rows correspond to the m cliques in G and whose columns correspond to n vertices, where m is the number of all the cliques in G and n is the number of all the vertices in G. Its (i,j)th element contains 1 if the ith clique contains the jth vertex, and 0 if not.

For a given M, its query graph G is defined as follows. In this definition, the union of two graphs means a graph which contain both of their vertices and edges.

Definition 6.9: Let G_i be a complete graph in which each vertex corresponds to each query and every two elements in $Q(r_i)$ is connected by an undirected edge. Then, the query graph of an RQ matrix, G is defined as $G= \cup G_i$. We assume that a vertex corresponding to a query q_j is also denoted by q_j in a query graph.

Figure 6.2 shows an example of an RQ matrix and Figure 6.3 shows its corresponding query graph.

Definition 6.10: For a given M, a set of m redundant queries is defined as

$$\{q_{n+1},\ldots,q_{n+m}\}$$

such that

$$R(q_{n+t})=\{r_t\}$$

for all t, $1\leq t\leq m$. An RQ matrix with a set of redundant queries added is denoted as M'. In a query graph G of M', a vertex

corresponding to a redundant query is called a redundant vertex and others are called non-redundant vertices.

	q_1	q_2	q_3	q_4
r_1	1	1	1	0
r_2	0	0	0	1
r_3	0	0	1	1
r_4	1	0	0	0
r_5	0	1	0	1

M

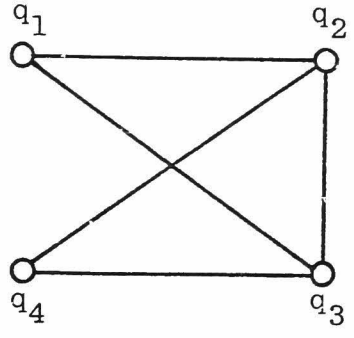


Fig.6.2. An example RQ matrix. Fig.6.3. The query graph of M in Fig.6.2.

As an example, M' of the RQ matrix M shown in Fig.6.2 and the query graph of the M' are shown in Fig.6.4 and Fig.6.5, respectively. In Fig.6.5, the nodes q_5, q_6, q_7, q_8 and q_9 are redundant vertices.

Definition 6.11: In G, any cycle corresponding to a row of M' is called an independent clique and others are called dependent cliques.

In the example shown in Fig.6.5, the clique which consists of q_1, q_2, q_3 and q_5 is an independent clique, but the one which consists of q_2, q_3 and q_4 is a dependent clique.

Definition 6.12: If every simple cycle of G , $v_1,\ldots,v_k$ for $k\geq 4$

has at least one triangular chord, G is called a chordal graph.

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9
r_1	1	1	1	0	1	0	0	0	0
r_2	0	0	0	1	0	1	0	0	0
r_3	0	0	1	1	0	0	1	0	0
r_4	1	0	0	0	0	0	0	1	0
r_5	0	1	0	1	0	0	0	0	1

M'

Fig.6.4. The M' of the RQ matrix M in Fig.6.2.

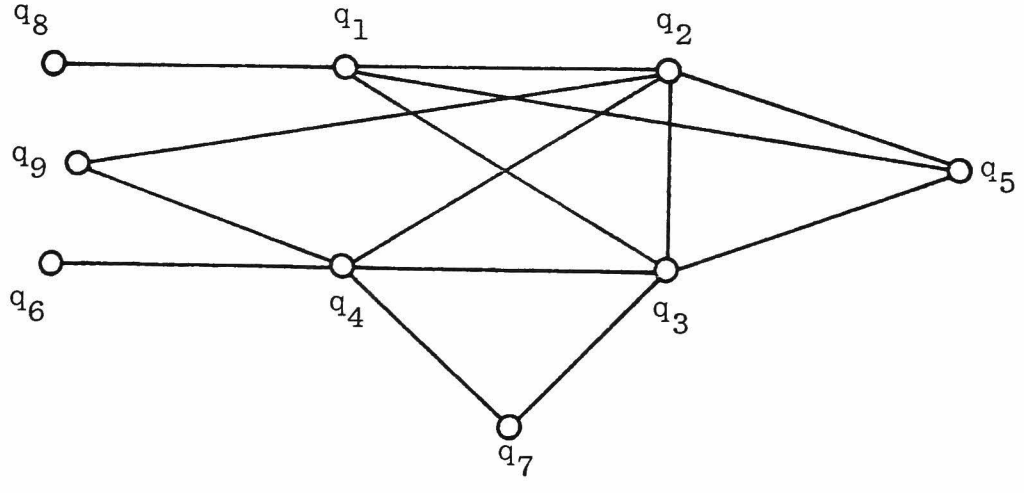


Fig.6.5. The query graph of M' in Fig.6.4.

6.3.2 Graph Theoretical Properties

In this subsection, we discuss a CR property of an RQ matrix from the viewpoint of graph theory, especially some relationships with interval graphs which have been studied by Fulkerson, Gross [FULKG65] and Gilmore, Hoffman [GILMH64]. Finally, a basic theorem to organize a QCR file without redundancy for a given buffer size is provided.

Theorem 6.1: [FULKG65]
An undirected graph G is an interval graph if and only if its clique vs vertex incidence matrix has the consecutive ones property for columns (the CR property) (proof omitted).

By this theorem, we immediately obtain the following lemma that relates an RQ matrix to an interval graph.

Lemma 6.1: A given RQ matrix has the CR property if it corresponds to the clique vs vertex incidence matrix of a certain interval graph (proof omitted).

A given RQ matrix does not necesarily correspond to a clique vs vertex incidence matrix of its query graph G. From the definition of the clique vs vertex incidence matrix, a given RQ matrix is a clique vs vertex incidence matrix of its query graph if and only if
(a) $Q(r_i) \not\supseteq Q(r_j)$ and $Q(r_i) \not\subseteq Q(r_j)$ for all i,j such that $1 \le i < j \le m$, and
(b) G does not contain any dependent clique.
In order to let a given RQ matrix M satisfy the condition (a), we introduce redundant queries and add them to M. In order to

make a given RQ matrix have a one-to-one correspondence with the clique vs vertex incidence matrix of its query graph, generally, it is not enough to add redundant queries to M. This occurs because there is a possibility that a dependent clique may be contained in the query graph of M' as shown in Fig.6.5. As for the existence of a dependent clique, the following lemma is obtained.

Lemma 6.2: An RQ matrix with the redundant queries added, M' has a CR property if and only if the RQ matrix M' is a clique vs vertex incidence matrix of a certain interval graph.

(Proof).· The sufficiency is obvious from lemma 6.1. As for the necessity, it is enough only to prove that the query graph of M' does not contain any dependent clique. Eswaran [ESWA7503] proved that whenever an RQ matrix has a CR property,

$$\bigcap_{j=1}^{t} R(q_j) \neq \emptyset$$

holds for any complete graph consisting of vertices q_1, q_2,...,q_t in the query graph G. Consequently, such a complete graph is always represented in the rows of M' which correspond to $\bigcap_{j=1}^{t} R(q_j)$. Since a clique is defined to be a maximal complete subgraph and both $Q(r_i) \not\supseteq Q(r_j)$ and $Q(r_i) \not\subseteq Q(r_j)$ hold, any clique in G corresponds to exactly one row of M'. Therefore, G does not contain any dependent clique. Q.E.D.

Lemma 6.2 guarantees that every clique in G appears as a row of M' if appropriate zero elements are rewritten into ones to make M' have a CR property. Conversely, even if appropriate zero elements are rewritten into ones so that all the dependent cliques may disappear, it does not necessarily guarantee the CR property of M'.

In Fig.6.6(a), an example of M' is shown which does not

have a CR property. Its query graph G is shown in Fig.6.6(b). G contains no dependent clique.

Gilmore and Hoffman characterized interval graphs in the following way.

(a)

	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8	q_9	q_{10}
r_1	1	1	1	0	0	0	1	0	0	0
r_2	0	1	0	1	1	0	0	1	0	0
r_3	0	0	1	0	1	1	0	0	1	0
r_4	0	1	1	0	1	0	0	0	0	1

M'

(b)

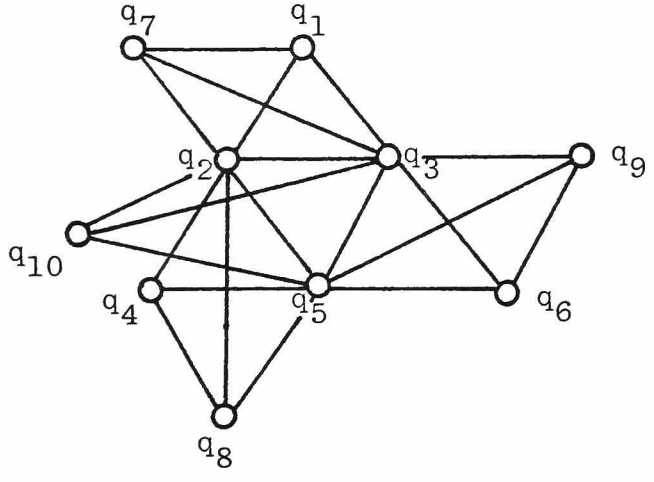


Fig.6.6. (a) An example of M' which does not have a CR property.
(b) The query graph of M' in Fig.6.6(a).

Theorem 6.2: [GILMH64]

A graph is an inetrval graph if and only if every quadrilateral in G has at least one diagonal and G^c is a comparability graph, that is, every odd cycle in the complement graph G^c has a triangular chord.

Hence, we can say that an RQ matrix M' has a CR property if and only if its query graph G is a chordal graph and G^c is a comparability graph [BOOTL7612]. Furthermore, we can relate a quadrilateral without any diagonal in G^c to an odd cycle without any triangular chord in a query graph G of M' in the following way.

Lemma 6.3: Let us assume that there does not exist any odd cycle without triangular chord in the complement graph G^c of a query graph G of M'. Then, G has no quadrilateral without diagonal.

(Proof). If there exists any simple cycle without any triangular chord in G, then such a cycle does not contain any redundant vertex. If such a cycle consists of four vertices, then this quadrilateral is covered with at least four independent cliques. Let us assume that G has a quadrilateral without any diagonal. Then, such a quadrilateral consists of four vertices each of which is a non-redundant vertex. And it is covered with at least four independent cliques. Then, G always contains a subgraph shown in Fig.6.7. Here, q_5, q_6, q_7 and q_8 are redundant vertices. In G^c of Fig.6.8, we can find an odd cycle without triangular chord; q_2, q_5, q_3, q_1, q_8, q_6, q_4, q_6, q_5. Then contradiction. Q.E.D.

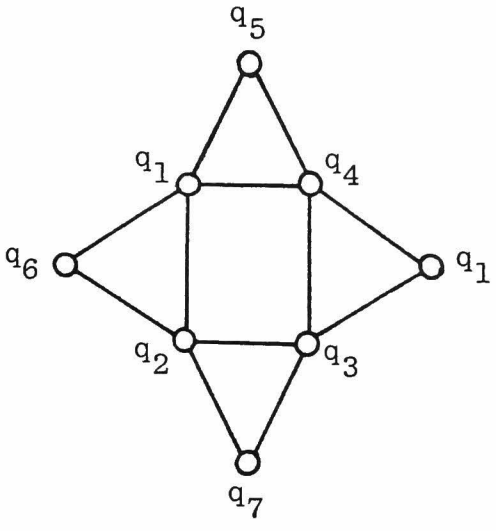


Fig.6.7. A quadrilateral without any diagonal which is contained as a subgraph of G.

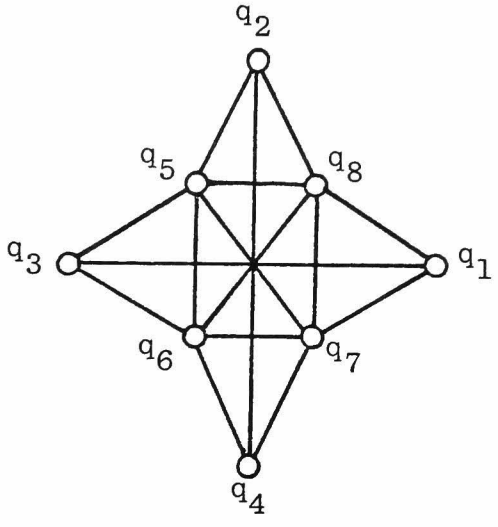


Fig.6.8. The complement graph of Fig.6.7.

From these discussions, since a given RQ matrix M has a CR property if and only if M' has a CR property, we can obtain the following theorem.

Theorem 6.3: A given query matrix M has a CR property if and only if the complement graph G^c of the query graph of M' is a comparability graph, that is, in G^c, every odd cycle has a triangular chord (proof omitted).

Let $d(q_j)$ be the number of all the edges each of which connects the non-redundant vertex q_j and a redundant vertex in G^c of M'. For each column of the given query matrix M, the

number of ones is represented in G^c as $m-d(q_j)$, where m is the number of all the records. As for the existence of a QCR file for a given buffer size, we obtain the following theorem by using Theorem 6.3 and $d(q_j)$.

Theorem 6.4: It is possible to organize a QCR file without redundancy for a given buffer size k if and only if there exists a set of edges in which edge connects a redundant vertex and a non-redundant vertex, such that the deletion of those edges can transform G^c into a comparability graph and does not cause $d(q_j)<m-k$ for any j ($1\leq j\leq n$) (proof omitted).

Fig.6.9 shows an example of G^c of M' which is also shown in Fig.6.4. We can find that the deletion of the edge (q_2,q_7) or (q_3,q_9) transforms the G^c into a comparability graph and that for every q_j, $d(q_j)\geq 2$ even after the deletion. Hence, we can organize a QCR file without redundancy for the buffer size $k\geq 3$.

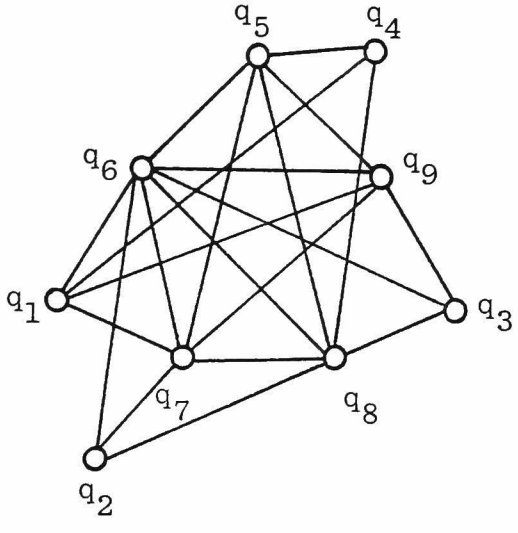


Fig.6.9. An example of G^c of M' in Fig.6.4.

In order to choose a candidate set of edges to be deleted, the algorithm developed by Pnueli et al. [PNUEL71] may be useful. It tests whether or not a given undirected graph is a comparability graph and also produces a transitively directed image of the graph under test.

Definition 6.13: The following definitions are given in [PNUEL71]. Let V and E be a set of vertices and a set of edges of G^c, respectively. A symmetric and irreflexive relation R on E is defined as follows: If $i \neq k$ and $(i,j),(j,k) \in E$, then $(i,j)R(j,k)$ if and only if $(i,k) \notin E$. Let G^{c^*} denote a partially oriented graph, that is, some (possibly all or none) of their edges are directed. An edge in G^{c^*} is called an implicant if it is directed and R-related to at least one undirected edge in G^{c^*}. A graph G^{c^*} is said to be stable if it contains no implicants; otherwise G^{c^*} is called unstable.

The following orientation rule is used in the algorithm. Choose an implicant edge [i,j]. Then, to every undirected edge (i,i') such that $[i,j]R(i,i')$ assign the direction i -> i', and to every undirected edge (j,j') such that $[i,j]R(j,j')$, assign the direction j' -> j.

Let us assume that the given G^c, that is, the complement graph of the query graph of M' is not a comparability graph. Let T be a set of all ordered pairs of directed edges (e_1,e_2) such that $e_1=[a,b]$, $e_2=[b,c]$, $e_1,e_2 \in E$ and $[a,c] \notin E$ for a,b,c in V. An edge e is said to cover a pair (e_1,e_2) if $e=e_1$ or $e=e_2$. Our basic approach to choose a candidate set of edges to

be deleted is as follows:

(a) Using the orientation rule introduced by Pnueli et al., direct all the edges in G^c. Whenever the resulting graph becomes stable, a new undirected edge is arbitrarily selected and the orientation rule is repeatedly applied.

(b) Find a set of edges by which any pair in T of the directed graph is covered.

Fig.6.10 shows one of directed images of G^c shown in Fig.6.9 using the method described above. In this case, at first, (q_4,q_5) is oriented as q_4 -> q_5. We can find that T= $([q_2,q_7], [q_7,q_1])$, $([q_2,q_7], [q_7,q_5])$. Since the directed edge $[q_2,q_7]$ covers T, G^c becomes a comparability graph in this case by its deletion. At this step, a total order over a set of redundant vertices is represented in this directed graph and we can obtain a QCR file, such as, $r_2r_5r_3r_1r_4$ according to the ordering. Of course, since there are many directed images of G^c, it is remained unsolved problem to develop an efficient algorithm for choosing the set of edges to be deleted, which satisfies the condition described in Theorem 6.4.

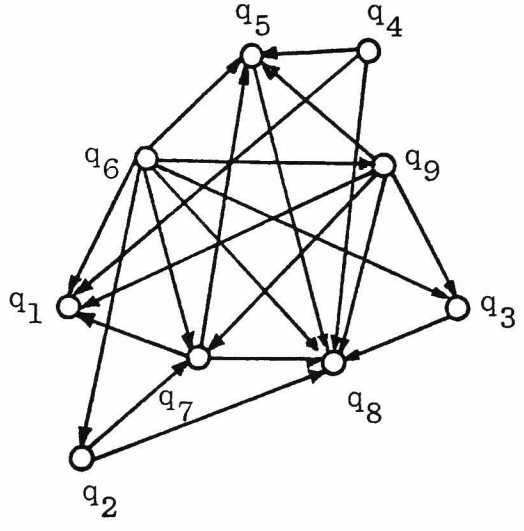


Fig.6.10. A directed image of G^c in Fig.6.9.

6.4 Organization of QCR Files with Reduced Redundancy

In this section, we discuss an organization which reduces the number of duplicates in a QCR file for a given buffer. Our approach for this problem consists of three stages, that is, to solve the preprocessing problem, the partition problem, and the placement problem. The latter two problems are solved by heuristic algorithms. The RQ matrix M shown in Fig.6.1 is used to illustrate several algorithms in this section. Detailed explanation using this example is provided in Subsection 6.4.4.

At the stage of solving the partition problem, the set of queries is partitioned into approximately minimized number of subsets. Assume that a given set of queries is partitioned into a family of subsets of queries $\{Q_1, Q_2, \ldots, Q_p\}$. For each i such that $1 \leq i \leq p$, R_i is defined to be a set of records each of which is pertinent to at least one query in Q_i. The number of records in R_i is limited to be less than or equal to a given buffer size for each i. Furthermore, we approximately minimize the total number of records in $R_i \cap R_j$ for all i,j such that $1 \leq i \neq j \leq p$. Here, for illustrative purposes, let k=5. For example, Q shown in Fig.6.1 is partitioned into Q_1, Q_2, Q_3 and Q_4, such that $Q_1=\{q_1, q_5\}$, $Q_2=\{q_2, q_8, q_9\}$, $Q_3=\{q_3, q_7\}$, and $Q_4=\{q_4, q_6\}$. By placing the records in R_1, R_2, R_3 and R_4 in this order, we obtain a QCR file for k=5 as follows:

$$r_1 r_2 r_3 r_4 r_7 r_2 r_3 r_4 r_5 r_6 r_3 r_5 r_6 r_7 r_1 r_3 r_5 r_7.$$

Note that in this example,

$$R_1=\{r_1, r_2, r_3, r_4, r_7\},$$
$$R_2=\{r_2, r_3, r_4, r_5, r_6\},$$

$R_3=\{r_3,r_5,r_6,r_7\}$,
$R_4=\{r_1,r_3,r_5,r_7\}$.

At this stage in solving the placement problem, first the initial arrangement of records is determined by using the solution obtained in the previous stage. At this step, the ordering of R_i's and the arrangement of records in each R_i are determined so that the number of overlapped records may become large for each pair of adjacent R_i and R_j. For the example solution obtained above, we obtain an initial arrangement of records, which is also a QCR file for a given buffer size k=5, as follows:

$$r_1r_7r_2r_3r_4r_5r_6r_3r_7r_1r_5.$$

Next, an operation called packing is used for this initial arrangement to delete some unnecessary duplicated records. In this example, note that we can further reduce the redundancy of this QCR file by the following operations; The fourth record r_3 and fifth record r_4 are interchanged. The sixth record r_5 and the seventh record r_6 are interchanged. The eighth record r_3 and the tenth record r_1 are deleted. Finally, we can obtain a QCR file with the redundancy 2 as follows:

$$r_1r_7r_2r_4r_3r_6r_5r_7r_5.$$

6.4.1 Preprocessing to Reduce the Size of the Problem

Let us assume that a set Q of queries is decomposed into a maximum number of subsets $Q_1,Q_2,\ldots,Q_t$ $(t\geq 2)$ such that $Q=\bigcup_{i=1}^{t}Q_i$, $Q_i\cap Q_j=\phi$, and $R_i\cap R_j=\phi$ for $i\neq j$, $1\leq i,j\leq t$. Here, a set of

records pertinent to Q_i is denoted by R_i. Then, the given problem can be decomposed into t problems of the same type each of which has a smaller size.

In order to determine whether or not this property is applicable to a given M and to find a maximum number of subsets if possible, we construct a graph G, regarding the RQ matrix M as the incidence matrix of a bipartite graph, and find all the connected components in G, each of which corresponds to a subset Q_i. When the number of vertices and edges of G are denoted by $n_v(=m+n)$ and e, respectively, an algorithm to find connected components is known, which requires $O(n_v^2)$ or $O(e)$ time.

By decomposing a set of queries as described above if possible, both the number of records and the number of queries are reduced.

6.4.2 Partition Problem

A partition of $Q=\{q_1,q_2,\ldots,q_n\}$ is defined as a family of subsets Q_j, $1\leq j\leq p$ such that $Q_i \cap Q_j=\phi$ for $i\neq j$, $1\leq i,j\leq p$ and $\bigcup_{j=1}^{p} Q_j=Q$. For every query q_j, $E(q_j)$ denotes $\sum_{j=1,j\neq i}^{n} |R(q_i) \cap R(q_j)|$. For every subset of queries Q_i, $S(Q_i)$ and $E(Q_i)$ denote $\left| \bigcup_{q_j \in Q_i} R(q_j) \right|$ and $\sum_{j\neq i}^{m} |R_i \cap R_j|$, respectively. In this chapter, $|\ |$ is used to denote the number of elements in a set.

A feasible partition is defined as a partition of Q such that $S(Q_i)\leq k$ (buffer size) for every i, $1\leq i\leq p$. The optimum solution of this problem is a feasible partition of Q such that the number of partition elements p are minimized with $\sum_{i}^{p} E(Q_i)$ also minimized.

Various algorithms to solve the partition problem defined

above are similar in many respects to those in the design automation, by which logic modules are partitioned onto boards with the number of interconnections approximately minimized. No efficient method is known for solving the partition problem which gives an optimum result and consequently, we use a heuristic algorithm introduced in [BREU73]. This algorithm solves the partition problem by solving an assignment problem iteratively after a so-called the 'seeding operation' is finished. In the seeding operation, the queries which are most likely to be cores of clusters, are planted into the partition elements as seeds. Therefore, all the partition elements grow simultaneously. In our algorithm, the initial number of partition elements is heuristically determined and if necessary, the number is increased one by one.

As for an organization of a QCR file with less redundancy, it is not easily estimated which partition algorithm is better. Because there is a possibility that two solutions of the partition problem may produce QCR files of the same length even if $\sum_i E(Q_i)$ of one solution is larger than that of another one. It depends on the solution of the placement problem described in Subsection 6.4.3.

A set of queries to be partitioned and its subset are denoted by Q and Q_0, respectively. The jth neighbourhood of a query q for $q \notin Q_0$ is denoted by $N_j(q)$ and defined as

$$N_1(q)=\{q\};$$
$$N_j(q)= \{q' \mid q' \in Q_0,\ |R(N_{j-1}(q)) \cap R(q')|>0\} \cup N_{j-1}(q)$$

for $j \leq 2$. Here, $R(N_{j-1}(q))$ denotes a set of records pertinent to queries in $N_{j-1}(q)$.

[Partition Algorithm]

(1) The number of partition elements p is assumed to be first unspecified. $Q_0:=Q$; $i:=1$; $j:=1$; Compute $R(q)$ for every q in Q_0;

(2) If $Q_0=\phi$, then $p:=i-1$ and go to (7); Select q in Q_0 such that $|R(q)|$ is a maximum; If there exists more than one candidate, select a query whose index is minimum; Assign q to the partition element Q_i and $Q_0:=Q_0-\{q\}$;

(3) If $Q_0-N_j(q)=\phi$ then $j:=j-1$ and go to (6); If $S(N_j(q))\leq k$ and $N_j(q)\neq N_{j-1}(q)$, then $j:=j+1$ and go to (3); If $S(N_j(q))>k$, then reconstruct $N_j(q)$ by adding each q' in Q_0, such that $|R(N_{j-1}(q)) \cap R(q')|>0$, in index order until $S(N_j(q))$ exceeds k for the first time, and go to (6); If $N_j(q)=N_{j-1}(q)$, then go to Step (6);

(4) If $Q_0=\phi$, then for all s such that $i\leq s\leq p$, $Q_s:=\phi$ and go to Step (7); Otherwise, select a query q in Q_0 such that $|R(q)|$ is a maximum. If there exists more than one candidate, select a query q whose index is a minimum. $Q_i:=Q_i \cup \{q\}$; If $i=p$, then go to (7); otherwise, $j:=1$ and go to (5);

(5) If $Q_0-N_j(q)=\phi$, then $j:=j-1$ and go to (5); If $S(N_j(q))\leq k$ and $N_j(q)\neq N_{j-1}(q)$, then $j:=j+1$ and go to (5); If $S(N_j(q))>k$ or $N_j(q)=N_{j-1}(q)$, then go to (6);

(6) $Q_0:=Q_0-N_j(q)$ and $i:=i+1$; If p is specified, then go to (4); otherwise go to (2);

(7) $Q_0=Q-\bigcup_{i=1}^{p} Q_i$;

(8) Let us assume that $Q_0=\{q_1,q_2,\ldots,q_{n'}\}$. Construct the correlation matrix $C=(c_{ij})$ in which each row corresponds to each partition element and each column corresponds to each query in Q_0, and the (i,j)th element of C is defined as $c_{ij}=|R_i \cap R(q_j)|$ for $1\leq i\leq p$, $1\leq j\leq n'$. If $S(Q_i \cup \{q_j\})>k$, then $-\infty$

is assigned to c_{ij}.

(9) If $c_{ij} \neq -\infty$ for every x_{ij} such that $x_{ij}=1$, then $Q_i := Q_i \cup \{q_j\}$ and $Q_0 := Q_0 - \bigcup_{i=1}^{p} Q_i$; If $Q_0 = \phi$, then this algorithm terminates; otherwise, if $Q_0 \neq \phi$ and $c_{ij} \neq -\infty$ for every x_{ij} such that $x_{ij}=1$ then go to (8); If there exists any c_{ij} such that $c_{ij}=-\infty$ and $x_{ij}=1$ then $p:=p+1$, $i:=1$, $Q_0:=Q$ and go to (4);

When this algorithm reaches (7), the seeding operation for p partition elements is finished. From this step, the assignment of queries remained is begun. At (9), the assignment variable x_{ij} is defined as follows: $x_{ij}=1$ in the case when a query q_j is assigned to Q_i; $x_{ij}=0$ otherwise; and $\sum_{j=1}^{m'} x_{ij}=1$. We obtain a solution of the asignment problem for which $\sum_{i,j} c_{ij}x_{ij}$ is maximized, using the correlation matrix C.

6.4.3 Placement Problem

We have obtained p sets of queries $Q_1, Q_2, \ldots, Q_p$ such that $|R_j| \leq k$ for $1 \leq j \leq p$ and both p and $\sum_{j=1}^{p} E(Q_j)$ are approximately minimized. The placement problem is to place all the records in R_j for all j on linear storage locations so that the number of records actually placed may be minimized. This problem is also solved by solving the following two subproblems.

First, the initial arrangement of records is obtained in the form shown in Fig.6.11. Records are assumed to be placed from left to right. In this figure, each subscript of a set of records is assumed to represent the order of those sets for their placement. Each part overlapped between R_{j-1} and R_j is denoted by R_j' for every j, $2 \leq j \leq p$. For every j, $1 \leq j \leq p$, R_j' is defined as follows:

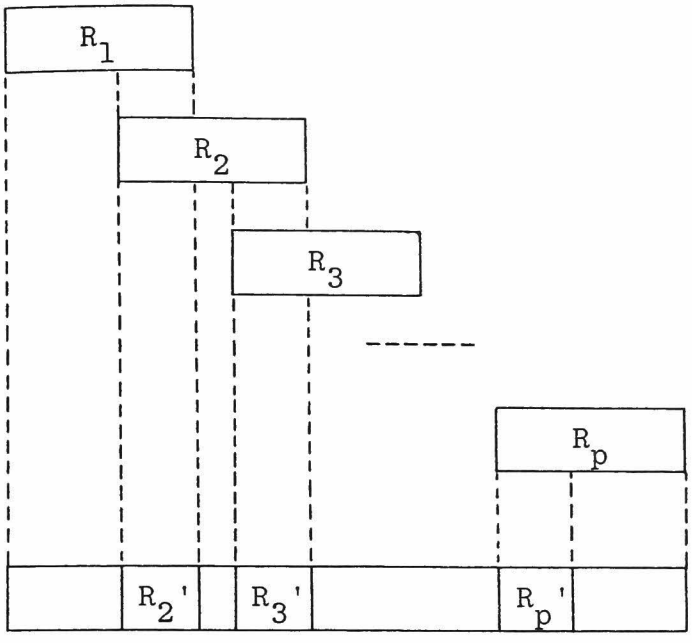


Fig.6.11. The initial arrangement of records.

$R_j' = \phi$ in the case of $j=1$;
$R_j' = R_1 \cap R_2$ in the case of $j=2$;
$R_j' = R_{j-1} \cap (R_{j-2} \cup R_j) - R_{j-1}'$ in the case of $3 \leq j \leq p$.

For every j, $1 \leq j \leq p-1$, obviously, we have

$$R_j' \cap R_{j+1}' = \phi .$$

The following algorithm decides the initial arrangement of records for given p sets of queries $Q_1, Q_2, \ldots, Q_p$. Let $R' = \{R_1, R_2, \ldots, R_p\}$ be a family of sets of records in which each R_i is a set of records corresponding to Q_i. In this algorithm, for any set A,B, A-B means a set of elements which only belongs to A, not to B.

[Initial Arrangement Algorithm]

(1) t:=1; $R_{j(t)}' := \phi$;

(2) Select R_x in R' such that $E(Q_x)$ is a minimum.

(3) j(t):=x; t:=t+1;

(4) If there exists R_y in R' such that $(R_{j(t-1)} - R_{j(t-1)}') \cap R_y \neq \phi$, then go to (6); otherwise, place all the records in $R_{j(t-1)}'$ and next, place all the records in $R_{j(t-1)} - R_{j(t-1)}'$;

(5) $R' := R' - \{R_{j(t-1)}\}$; If $R' = \phi$, then terminate; otherwise, $R_{j(t-1)}' := \phi$ and go to (2);

(6) Select R_y such that $|R_y \cap (R_{j(t-1)} - R_{j(t-1)}')|$ is a maximum; j(t):=y;

(7) Place all the records in $R_{j(t-1)}'$ and next, place all the records in $R_{j(t-1)} - R_{j(t-1)}' - R_{j(t)}$; $R_{j(t)}' := (R_{j(t-1)} - R_{j(t-1)}') \cap R_{j(t)}$;

(8) $R' := R' - \{R_{j(t-1)}\}$; t:=t+1; go to (4);

At step (2) and Step (6), when there exists more than one candidate to be selected, it is assumed to be selected in an index order.

Next, some unnecessary duplicate records are deleted by the packing operation described below. The initial arrangement of records shown in Fig.6.11 is an input for the packing algorithm. It should be noted that for every i, $1 \leq i \leq p-1$, $R_i' \cap R_{i+1}' = \phi$ and that for every i, $1 \leq i \leq k-2$, $(R_i - R_i' - R_{i+1}') \cap R_{i+2}' = \phi$. If there exists any record r which is commonly contained in these two sets, then r should be contained in R_i and R_{i+1}. By the definition, r should be placed in R_{i+1}'. However, $R_{i+1}' \cap R_{i+2}' = \phi$. Thus contradiction. On the contrary, it should be noted that $R_i' \cap (R_{i+1} - R_{i+1}' - R_{i+2}')$ is not necessarily

an empty set. Using these properties, the packing algorithm is constructed as follows:

[Packing Algorithm]

(1) i:=1;

(2) If $R_i' \cap (R_{i+1}-R_{i+1}') \neq \phi$, then go to (3); If $i \neq p$, then i:=i+1 and go to (2); otherwise, terminate;

(3) Rearrange all the records in R_i' into the followings: First, place all the records in $R_i'-R_i \cap (R_{i+1}-R_{i+1}')$; Second, place all the records in $R_i' \cap R_{i+2}'$; Finally, place all the records in $R_i' \cap (R_{i+1}-R_{i+1}'-R_{i+2}')$; In each part, records are placed in an index order;

(4) If $R_i' \cap (R_{i+1}-R_{i+1}'-R_{i+2}')=\phi$, then $B_1:=\phi$ and go to (5); otherwise, $B_1:=R_i' \cap (R_{i+1}-R_{i+1}'-R_{i+2}')$;

(5) If $R_i' \cap R_{i+2}'=\phi$, then $B_2:=\phi$ and go to (6); Otherwise, $B_2:=R_i' \cap R_{i+2}'$;

(6) If $|R_{i+1}|+|R_i-R_i'-R_{i+1}'| \leq k$, then go to (7); Otherwise, go to (10);

(7) Delete the records in B_1 and B_2 from R_{i+1};

(8) Pick up k records consecutively placed from right to left in the area which contains all the records in R_{i+2}' as exactly the rightmost part; Examine whether these k records contain all the records pertinent to Q_{i+1}; If no pertinent record lacks, then go to (10); otherwise, add B_1 to $R_{i+1}-R_{i+1}'-R_{i+2}'$, and add B_2 to R_{i+2}';

(9) Pick up k records consecutively placed from right to left in the area which contains R_{i+2}' as exactly the rightmost part. Add the lacking records, which are also pertinent to Q_{i+1} and go to (6);

(10) i:=i+1; $B_1:=\phi$; $B_2:=\phi$; Go to (2);

6.4.4 Example of a QCR File with Redundancy

In Fig.6.1, an example of a query matrix M is shown. A buffer is assumed to be able to contain at most five records. This example is used to illustrate the organization algorithm of a QCR file with redundancy described in this section.

(1) Preprocessing

For this query matrix, since there exists neither partition of queries to reduce the size of problem nor identical columns, this operation is not done.

(2) Partition of Queries

Let us solve the partition problem for the Q of this example by the algorithm shown in 6.4.2. First, select q_1 and $Q_1=\{q_1\}$, $N_0(q_1)=\{q_1\}$, $N_1(q_1)=\{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9\}$, and $S(N_1(q_1))=7>k$. Thus, $N_1(q_1)=\{q_1,q_2,q_3\}$ and $Q_0=Q_0-N_1(q_1)$. Similarly, $Q_2=\{q_4\}$ and $N_1(q_4)=\{q_4,q_5,q_6,q_7\}$, $Q_3=\{q_8\}$, $N_1(q_8)=\{q_8,q_9\}$. Then p=3 and the seeds q_1, q_4 and q_8 are assigned to Q_1, Q_2 and Q_3, respectively. The correlation matrices are shown in Fig.6.12(a) and (b). In this case, it proves to be impossible to partition $Q_0=\{q_2,q_3,q_5,q_6,q_7,q_9\}$ into three partition elements. Then, p:=p+1. In this case, the seeds q_1, q_2, q_3 and q_4 are assigned to Q_1, Q_2, Q_3 and Q_4, respectively. The correlation matrix is shown in Fig.6.12(c). Then, $Q_1=\{q_1,q_5\}$, $Q_2=\{q_2,q_8\}$, $Q_3=\{q_3,q_7\}$ and $Q_4=\{q_4,q_6\}$. Finally, we obtain the following solution.

$Q_1=\{q_1,q_5\}$,
$Q_2=\{q_2,q_8,q_9\}$,
$Q_3=\{q_3,q_7\}$,
$Q_4=\{q_4,q_6\}$.

(a)

	q_2	q_3	q_5	q_6	q_7	q_9
$Q_1=\{q_1\}$	2	$-\infty$	2	$-\infty$	$-\infty$	1
$Q_2=\{q_4\}$	2	2	1	2	2	0
$Q_3=\{q_8\}$	2	2	0	1	1	1

(b)

	q_5	q_7	q_9
$Q_1=\{q_1, q_2\}$	$-\infty$	$-\infty$	$-\infty$
$Q_2=\{q_4, q_6\}$	2	2	$-\infty$
$Q_3=\{q_3, q_8\}$	$-\infty$	2	2

(c)

	q_5	q_6	q_7	q_8	q_9
$Q_1=\{q_1\}$	2	$-\infty$	$-\infty$	2	1
$Q_2=\{q_2\}$	1	1	1	2	0
$Q_3=\{q_3\}$	$-\infty$	1	2	2	1
$Q_4=\{q_4\}$	1	2	2	2	0

Fig.6.12. Examples of Correlation matrices.

(3) Placement of Records

(3-1) Initial Arrangement of Records

From (2), we have obtained $R'=\{R_1, R_2, R_3, R_4\}$ such that

$R_1=\{r_1,r_2,r_3,r_4,r_7\}$,
$R_2=\{r_2,r_3,r_4,r_5,r_6\}$,
$R_3=\{r_3,r_5,r_6,r_7\}$, and
$R_4=\{r_1,r_3,r_5,r_7\}$.

According to the initial arrangement, we have a sequence of records of length 11, $r_1r_7r_2r_3r_4r_5r_6r_3r_7r_1r_5$ which is obtained from the result j(1)=1, j(2)=2, j(3)=3 and j(4)=4.

(3-2) Packing Operation

In Fig.6.13, we show the initial arrangement of records and the final arrangement of records obtained by the packing operation. The broken line denotes an area in which each R_i is placed.

Initially, $R_1'=\phi$, $R_2'=\{r_2,r_3,r_4\}$, $R_3'=\{r_5,r_6\}$ and $R_4'=\{r_3,r_7\}$. In this operation, since $R_2' \cap (R_3-R_3')\neq\phi$, R_2' is rearranged as $r_2r_4r_3$ and r_3 is deleted from R_3'. Besides, since $R_3' \cap (R_4-R_4')\neq\phi$, R_3' is rearranged as r_6r_5 and r_5 is deleted from R_4. Finally, we obtain a QCR file with two duplicated records, $r_1r_7r_2r_4r_3r_6r_5r_7r_5$. In Fig.6.13, the bidirectional arrow denotes each area containing $R(q_j)$. In this case, the optimal solution is not obtained. One of QCR files with minimum redundancy for this RQ matrix is, for example, $r_6r_7r_4r_3r_5r_1r_2r_7$.

6.5 Concluding Remarks

In this chapter, we have introduced a new file organization called the QCR file organization. In this organization, both the retrieval time and the amount of storage space are reasonably reduced.

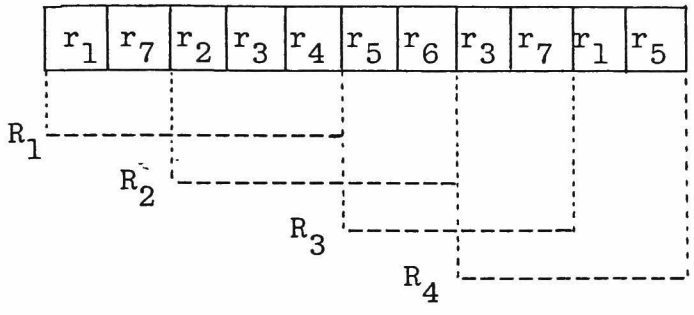
We obtained graph theoretical properties on the CR property introduced by Ghosh and a basic theorem to organize a QCR file

without redundancy for a given buffer size.

By using redundant queries, several properties concerned with interval graphs are simplified.

It is important to organize a QCR file with reduced redundancy, and we have provided an organization algorithm for such a file.

Initial Arrangement of Records



Final Arrangement of Records Obtained by Packing Operation

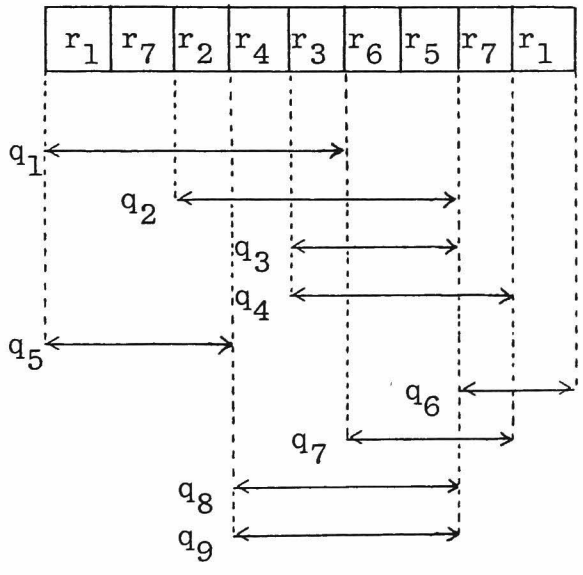


Fig.6.13. An example of the placement problem.

CHAPTER 7 RELATIONAL INVERTED STRUCTURE FILES

In this chapter, a new access path structure called Relational Inverted Structure (for short, RIS) supporting relational database operations efficiently is introduced. The design of RIS files is aimed at increasing the efficiency of processing relational operations and balancing the required response time of relational operations on any attribute. We introduce the notion of 'pseudo' relational operations by using hashed attribute values in RIS files. The hashed attribute values are used to decrease the number of accessions to a relation by eliminating tuples which can be proved not to be contained in the answer in advance. The information of functional dependencies is also used to increase the efficiency of query processing in RIS files. Actually, RIS files are adopted as the access path structure of the research information system ARIS, which have been developed at Yajima Laboratory in Kyoto University.

7.1 Introduction

In the relational data model, since all information is represented by data values, there is no 'preferred' format for a query at the user interface [CHAM7602]. That is, users' queries are logically symmetric in the relational data model. It is, however, not necessarily guaranteed that the response time of various queries are balanced. Besides being important, these concepts are very difficult to realize because their realization is also concerned with many other factors, such as query

evaluation techniques and maintenance techniques of access path structures after update operations.

In this chapter, a new access path structure called Relational Inverted Structure (for short, RIS) is presented. RIS is a generalization of Haerder's 'Generalized Access Path Structure' [HAER7610]. A record of a RIS file consists of a domain value, relation number, attribute number, a list of tuple identification codes (for short, TIDs) and a list of hashed values. RIS files are designed to achieve physical symmetry as much as possible at the attribute, operation and maintenance level.

The main characteristics of RIS files are as follows:

(1) Conventional access paths, such as links and secondary indexes, can be realized by RIS files.

(2) Selection, join and division operations in relational databases are efficiently processed by 'pseudo' operations using hashed attribute values. The hashed values are used to decrease the number of accessions to a relation by eliminating TIDs which are obviously proved not to be contained in the answer in advance.

(3) The information of functional dependencies is used to process queries efficiently.

(4) Integrity constraints such as functional dependencies and some interrelation constraints can be checked using RIS files only.

(5) View handling, especially the checking facility of user's view update viability, is provided.

In this chapter, we mainly discuss the characteristics (1), (2) and (3) of RIS files. Detailed description of RIS files and other characteristics are provided in [TANAK7808] [LEVII8003].

In order to support accessing to data, many access path

structures have been proposed. These access paths are managed and updated appropriately according to changes of data. For simple maintenance and implementation, these access path structures must be implemented as a unified structure.

Two kinds of access paths, secondary indexes and links (pointer chains) have been presented and implemented in System R [ASTRB7606]. Secondary index provides associative access capability, which is useful to retrieve a set of tuples whose some attribute values are the same. Link is used to connect physically tuples in one or two relations. In order to decrease the implementation complexity of these two access paths, Haerder proposed a generalized access path structure, which combines the advantages of secondary indexes and links [HAER7610]. However, division operation in relational databases is not sufficiently supported by either secondary indexes, links or Haerder's access paths. Moreover, not all attributes have indexes because in the case of full inversion (all the attributes are indexed on) maintenance of these access paths is very complicated and costly. Thus these access paths seem weak with respect to queries involving attributes which have no indexes.

A RIS file provides a secondary index for the set of 'domain-related' attributes using TIDs. Hashed values are also incorporated into RIS file records in order to support division operation and queries on attributes which are not indexed on. Usually, not all attributes are provided with a RIS file in order to decrease the costs of update maintenance. The cost of update maintenance in RIS files is considered to be bounded by the update maintenance cost in full inversion systems. Another trade-off paid for RIS files is the storage space required for hashed values of attribute values.

Basic definitions are given in Section 7.2. In

Section 7.3, the RIS file organization and processing algorithms for basic relational operations are provided. In Section 7.4, efficient processing algorithms for combined relational operations are provided. In addition, we also show the usage of functional dependencies in processing certain types of compound relational operations.

7.2 Basic Definitions

Given two attributes A_i and A_j, the domains of A_i and A_j are denoted by $DOM(A_i)$ and $DOM(A_j)$, respectively. The term domain-related is defined as follows:

(1) If $DOM(A_i) \cap DOM(A_j) \neq \phi$ then A_i and A_j are domain-related, denoted $A_i \sim A_j$.

(2) For distinct i, j and k, if $A_i \sim A_j$ and $A_j \sim A_k$, then $A_i \sim A_k$.

Related to the semantic aspects of domains, a database administrator may divide a set of domain-related attributes into smaller sets of attributes, and for each such set, a RIS file can be constructed.

The selection operator in relational algebra selects only those tuples of a relation which satisfy a given condition on an attribute. Formally, let θ be any element in $\{=,=,<,<=,>,>=\}$ and A be an attribute of a relation r. The θ-selection of r on attribute A is defined by

$$r[A\theta'c'] = \{t: t \in r, t[A]\ \theta\ 'c', c \text{ is a constant}\}.$$

Here, t[A] denotes the value of the attribute A of tuple t. A special but often used case, where θ='=', is simply called a selection.

The division operator is useful in expressing queries which contain a universal quantifier of the relational calculus. Consider two relations r on R= A $\bar{A}$ and s on S= B $\bar{B}$. The division of r on A by s on B results in a relation r' on $\bar{A}$ consisting of $\bar{A}$ values, each of which has all B-values on A in relation r. $\bar{A}$ means the complement of A in the set R of attributes. The division of r_1 on A by r_2 on B is defined by

$$r_1[A \div B] r_2 = \{t[A] : t \in r_1,\ r_2[B] \subseteq r_1[t[A], A]\}.$$

7.3 RIS File Organization and Basic Query Processing

7.3.1 RIS File Organization

Usually, two kinds of accesses are required in relational database management systems: (1) the direct access to a certain tuple in a relation and (2) the navigational access from tuples to tuples of different relations. It is assumed that each tuple in a relation is uniquely determined by the tuple identification code (for short, TID). Accesses to individual tuples are carried out through the TIDs.

The first kind of access can be implemented by (secondary) indexes on attribute values. Each attribute value is associated with a list of TIDs whose corresponding tuples have the attribute value. Processing of selection is really sped up if such an index exists for attributes in queries. The second one is represented by the connection of TIDs whose corresponding

tuples match on some attribute values, which is usually implemented as links.

The RIS file organization has the following characteristics: (1) Domain-based storage structure, and (2) Using hashed values to perform pseudo operations shown in this chapter.

The following rules can be adopted as criteria of selecting attributes on which a RIS file is constructed. A RIS file is constructed for

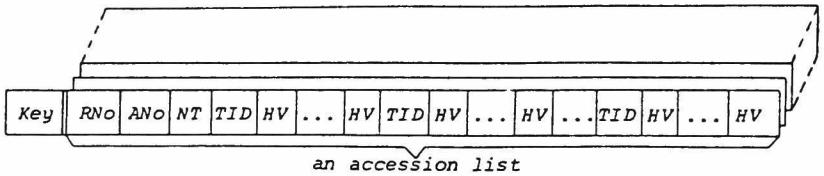
(1) a set of domain-related attributes, furthermore

(2) a key attribute (this is needed for checking the duplication of key values when tuples are inserted.)

(3) an attribute with high access frequency.

The RIS file for a given set of domain-related attributes $\{A_{i1},\ldots,A_{ik}\}$ is denoted by $RIS(A_i)$. Hereafter, the term 'domain value' x in $RIS(A_i)$ means a value which belongs to at least one $DOM(A_{ij})$ $(j=1,\ldots,k)$.

A record according to a domain value in a RIS file consists of the domain value and a number of accession lists. Each accession list corresponds to a relation and an attribute, and has a list of TIDs whose corresponding tuples own the domain value. Incorporated into each TID is a list of hashed values of attributes other than the attribute A_{ij} in the relation. Fig.7.1 illustrates the structure of a RIS file record. For a given sample data in Fig.7.2, an example illustrating a RIS file is presented in Fig.7.3.



RNo: Relation number

ANo: Attribute number

HV: Hashed value

NT : Number of tuples

Fig.7.1. Structure of a RIS file record.

r_1	A	B	C
#1	1	a	x
#2	1	b	y
#3	1	c	z
#4	2	a	x
#5	2	d	u
#6	2	b	y
#7	3	a	x
#8	3	b	v

r_2	D	B	E
#1	p	a	x'
#2	p	b	y'
#3	p	c	z'
#4	p	d	u'
#5	q	a	x'
#6	q	b	v'

Fig.7.2. A sample date.

a	r_1	B	3	#1	#4	#7	r_2	B	2	#1	#5
				h(1)	h(2)	h(3)				h(p)	h(q)
				h(x)	h(x)	h(x)				h(x')	h(x')

b	r_1	B	3	#2	#6	#8	r_2	B	2	#2	#6
				h(1)	h(2)	h(3)				h(p)	h(q)
				h(y)	h(y)	h(v)				h(y')	h(v')

c	r_1	B	1	#3	r_2	B	1	#3
				h(1)				h(p)
				h(z)				h(z')

d	r_1	B	1	#5	r_2	B	1	#4
				h(2)				h(p)
				h(u)				h(u')

h(x): hashed value of x

Fig.7.3. The RIS file records of attribute B.

7.3.2 Processing of Basic Relational Operations

(A) Selection

Consider a relation r on $R = A\bar{A}$, where $\bar{A}$ is the complement set of A in R. Suppose that a sample selection Q in Fig.7.4(a) is given. Depending on the existence of the RIS file for attribute A in the query (RIS(A)), possible access paths are grouped into two classes called P1 and P2 as shown in Fig.7.4(b). Here,

P1: RIS(A) is accessed directly through the specified value R.A='c', and the set of corresponding TIDs are obtained. Using these TIDs, corresponding tuples in r are accessed.

P2: RIS(A'), where $A' \in \bar{A}$, is accessed sequentially. The TIDs which have h(A)=h('c') are only output. Here, h(A) and h('c') mean the hashed value of attribute A of the corresponding tuple and the hashed value of 'c', respectively.

The method in P1 is clearly more efficient than the one in P2 because of the sequential access mode in P2. The flow chart given in Fig.7.4(b) helps us in decision of which access path to be used. The operation corresponding to P2 is called the pseudo selection, which is performed not on real data values, but on hashed attribute values. An access path decision table is given in Fig.7.4(c). For implementation, access paths are numbered so that when more than one access path are possible, the one with the smallest number is the most efficient one. However, it is noted that depending on the size of relation files and RIS files, there is some case where scanning relation files is more efficient than scanning RIS files.

(B) Join

Similar to selection, there are two possible classes of access paths for join operations. Given two relations r_1 on R_1 and r_2 on R_2 and a query Q in Fig.7.5(a) (here, A and B must be domain-related attributes), the access path decision flow chart and the decision table are given in Fig.7.5(b) and (c), respectively. Possible access paths according to the existence of RIS files for attribute of R_1 and R_2 are explained as follows:

P1: RIS(A) is sequentially accessed and TIDs of r_1 and r_2 in each RIS file record are output. Using these pairs of TIDs, relations r_1 and r_2 are accessed directly and corresponding tuples in the two relations give the correct result of the join operation.

P2: In this case, RIS(A'), where $A' \in \bar{A}$, is equal to RIS(B'). Thus, RIS(A') is accessed sequentially and the pairs (r_1.TID, r_1.h(A)) and (r_2.TID, r_2.h(B)) are output. Here, r_i.TID means a TID of a tuple in r_1 and r_i.h(X) means a hashed value of X-value of the corresponding tuple. These pairs are tested whether the hashed values match. Only those pairs such that the hashed values match are used to fetch the corresponding tuples from relations. After fetching the actual tuples, only a pair of tuples such that the actual attribute value match are output as the result.

P3: RIS(A') is accessed sequentially and a set of (r_1.TID, r_1.h(A)) pairs are output. Next, RIS(B') is also accesses sequentially to obtain a set of (r_2.TID, r_2.h(B)) pairs. The processing after this is the same as the one in P2.

P4: The relation files of r_1 and r_2 are accessed sequentially to perform the join operation. Tuples from r_1 and r_2 are actually compared to get those satisfying the join

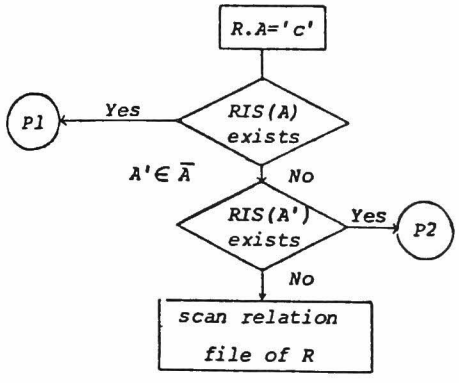


Fig.7.4(b) Access path decision flow chart.

$R = \{A, \bar{A}\}$

$Q: R.A = 'C'$

Fig.7.4(a) A sample qualification with selection.

	A	A'∈ Ā
A	P1	P1
A'∈ Ā	P1	P2

Fig.7.4(c) Access path decision table.

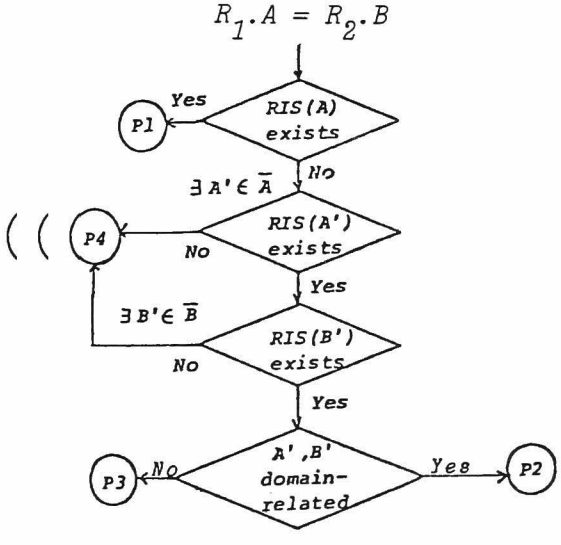


Fig.7.5(b) Access path decision flow chart for join operation.

$R_1 = \{A, \bar{A}\}$

$R_2 = \{B, \bar{B}\}$

$Q: R_1.A = R_2.B$

Fig.7.5(a) A qualification with join operation.

	A'∈ Ā	A,B	B'∈ B̄
A'∈ Ā	P3	P1	P2,P3
A, B	P1	P1	P1
B'∈ B̄	P2,P3	P1	P3

Fig.7.5(c) Access path decision table for join operation.

qualification.

The operation of P2 and P3 is called a pseudo join operation.

(C) Division

Consider again relation r_1 on R= A $\bar{A}$ and r_2 on R_2= B $\bar{B}$ and a query Q: $r_1[A \div B]r_2$. Four possible access paths are possible. Access path flow chart and the decision table are the same as those of join operation.

Here, only the case when RIS(A) (=RIS(B)) exists, is explained. First RIS(A) is sequentially accessed to obtain a set of (r_1.TID, r_1.h(A)) pairs in each RIS record. Based on the definition of division, when we find a RIS record with TIDs of r_2 but without any TID of r_1, we can conclude that the result of the division is empty. Any RIS record without TIDs of r_2 can be disregarded. Each set of (r_1.TID, r_1.h(A)) pairs corresponds to each B-value in relation r_2. Therefore, the resulting TIDs are those with h(A) appearing in all the sets. Here, because hashed values of A are used instead of the values themselves, final check on these values is needed. Note in this case, the division operation is performed as efficiently as the case of join by using hashed values.

7.4 Processing Algorithms for Compound Operations

In this section, some efficient algorithm for multiple relational operations (called compound operations) are described. In 7.4.1, these algorithms are described for (1) a

combination of the same type of operations, and (2) a combination of different types of operations, separately. In 7.4.2, as a special case of (1), further efficient algorithms using Codd's functional dependencies are described

7.4.1 Processing of Compound Operations

(A) Combination of Same type of Operations

In Section 7.3, processing algorithms for basic relational operations are shown. These algorithms can be combined to be used for processing compound operations. But, we note that a compound operation can be further efficiently processed as explained below.

In the case of a combination of join operations, the following algorithm is applied:

Assume that we process a query: $R_1.A=R_2.A'$ and $R_1.B=R_2.B'$.
(1) If there exists neither RIS(A) (=RIS(A')) nor RIS(B) (=RIS(B')), then the processing algorithm follows the case of P2 or P3 shown in the join processing algorithm in 7.3.2.
(2) If there exists RIS(A) (=RIS(A')) or RIS(B) (=RIS(B')), one of them, for example, RIS(A) is selected.
(3) RIS(A) is sequentially accessed to obtain a set of (r_1.TID, r_2.TID) each of which is associated with the same domain value and h(B)=h(B') in RIS(A).
(4) Relations r_1 and r_2 are accessed by these pairs of TIDs, and are checked for each pair whether or not the actual value on B of r_1 is equal to the value on B' of r_2. If they are the same, then this concatenation of the tuples is selected as a result.

In the case of selection operations, two possibilities occur. That is, the case when both RIS(A) and RIS(B) are used if exist, and the case when one of them is used. In the former case, it is necessary to access directly two RIS files RIS(A) and RIS(B) to obtain the intersection of two sets of TIDs. Here, we consider a query such that R.A='a' and R.B='b'. In the latter case, it is necessary to access directly one RIS file (e.g., RIS(A)) and to obtain a set of (r_1.TID, h(B)) pairs, each of which has the domain value 'a' and h(B)=h('b'). Then, the relation r_1 is accessed by these TIDs, and the obtained tuples are checked to select only tuples having the B-value 'b' actually.

(B) Combination of Different Types of Operations

In general, the qualification of a query is a combination of different types of relational operations. The simplest way to treat this problem is to evaluate each operation independently of the other ones yielding a set of TIDs for each operation. The overall result is included in these individual results. In fact, this method is not lead to an optimal performance. Because of the hashed attribute values in RIS files, more efficient algorithms are available.

Basically, three cases are considered: join-selection, selection-division and join-division. For further complicated operations, a generalization of these cases can be considered. Here, the former two cases are explained using examples.

(B-1) Join-Selection

Consider relations r on R=ABC and s on S=BD and a query in Fig.7.6. First, RIS(C) is directly accessed by the domain value 'c' to obtain a set of pairs (r.TID,r.h(B)). Next, RIS(D) is also directly accessed by D-value 'd', and a set of pairs (s.TID, s.h(B)) are obtained. The obtained TIDs of r and s, which match on h(B), are selected as pseudo-results. Finally, relations r and s are read in for checking.

(B-2) Selection-Division

Consider relations r_1 on R_1= A $\bar{A}$ and r_2 on R_2= B C $\overline{BC}$ and a query Q: $r_1[A \div B](r_2[C='c'])$, where $r_2[C='c']$ denotes a selection operation, which results in a subrelation of r_2 in which the C-value of every tuple is 'c'. Here, values of divisor B are restricted corresponding to the specified constsnt 'c'. In this case, many possibilities exist. The access path decision flow chart and the decision table are given in Fig.7.7. Here, only the case of P1 in Fig.7.7 is explained. Other cases are described in [TANAL7808].

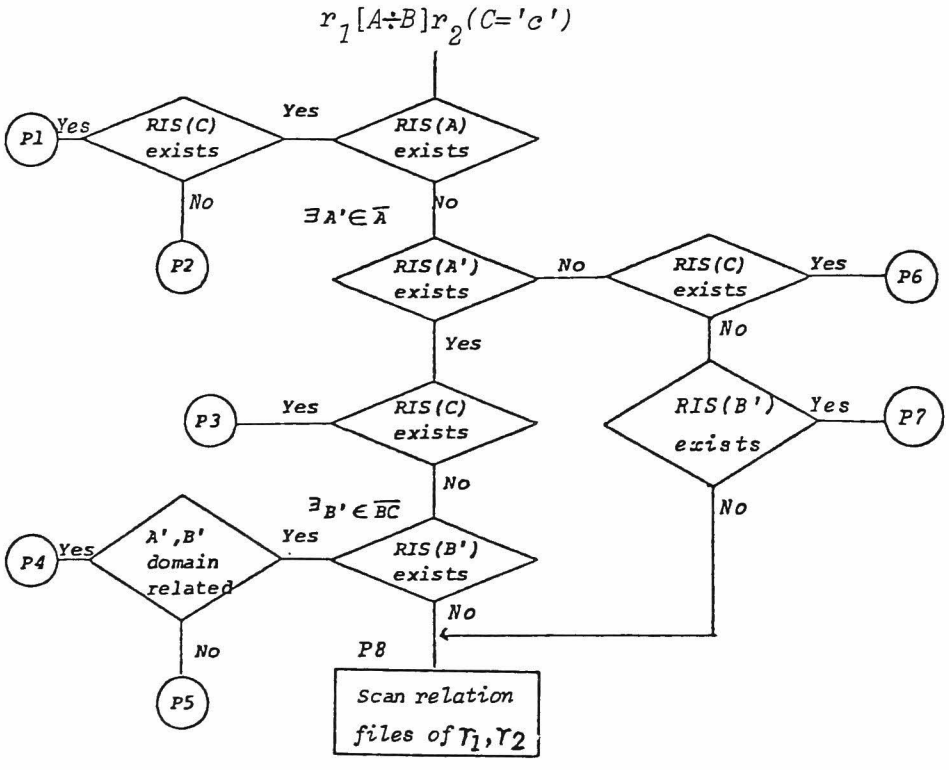
(1) First, RIS(C) is directly accessed by the domain value 'c' to obtain a set of TIDs of r_2.
(2) The relation r_2 is directly accessed by the set of TIDs obtained in (1).
(3) RIS(B) (=RIS(A)) is directly accessed by B-values obtained in (2) and for a set of RIS(B) records, the same algorithm for division as described in Section 7.3.2 is applied.

7.4.2 Query Processing Using Functional Dependencies

Assume that the functional dependency X -> Y holds in

```
GET     (R.A)
WHERE   R.C = 'c'
AND     R.B = S.B AND S.D = 'd'
```

Fig.7.6. A sample query with join and restriction.



	$A = B$	$A' \in \overline{A}$	C	$B' \in \overline{BC}$
$A = B$	P2	P2	P1	P2
$A' \in \overline{A}$	P2	P8	P3	P4, P5
C	P1	P3	P6	P6
$B' \in \overline{BC}$	P2	P4, P5	P6	P7

Fig.7.7. The access path decision flow chart and table for division.

relation r_1. Obviously, the functional dependency X -> H(Y) also holds, where H(Y) is a set of hashed values of Y-values. Using this simple idea and RIS files, basic relational operations can be further efficiently processed as follows:

(A)Combination of Selection Operations

Consider the relation r on R=ABC where A and B are keys of R. Let us consider the processing of a query r[B='b'] $\bigcap$ r[C='c']. Processing of this type of operation has already been described in Section 7.4.1, however, functional dependencies are useful to deciding which RIS file (RIS(B) or RIS(C)) is to be selected as follows.

Here, since B is a key, B -> AC holds in r. That is, for each B-value, there is at most one AC-value. In fact, if RIS(B) is chosen to be read, the resultant TID set consists of only one TID. This is usually not true for the case of C. Hence, as long as B -> AC holds, the access from RIS(B) always results in one TID. Therefore, it is better to choose RIS(B) because not only searches of general case but also unsuccessful searches are fast achieved.

(B) Division

Consider relations r_1 on R_1= B $\overline{B}$ and r_2 on R_2= C $\overline{C}$ and the division of r_1 on B by r_2 on C ($r_1[B \div C]r_2$). For discussion purpose, r_1 and the projection $r_2[C]$ are called dividend relation and divisor relation, respectively. Following the definition of division, a B-value of r_1 is a result value if the set of B-values of r_1 corresponding to the B-value contains the set of C-values of r_2. If all B-values of r_1 do not satisfy

this condition, the result is an empty set. The response in this case is a negative one. A search result in a negative response is called an unsuccessful search. An unsuccessful search usually takes a lot of time because it needs a search of all relations concerned.

An unsuccessful search is sped up in the following case. If the FD $\overline{B}$ -> B holds in the dividend relation, the set of B-values of r_1 corresponding to each B-value of r_1 has only one element. Provided this functional dependency, if the set of C-values of r_2 has more than one element, then the result is empty. Therefore, a division with the dividend relation, in which the FD $\overline{B}$ -> B holds, can be efficiently processed by first checking the number of values in the divisor relation. If the number is greater than 1, no more redundant operations are needed, and the division results in an empty set.

7.5 Concluding Remarks

In this chapter, a new access path structure called RIS supporting relational database operations efficiently is introduced. The main results of this chapter are (1) a new unified access path structure with 'pseudo' relational operation processing capability, which is suitable for selection, join and division operations, (2) query processing algorithms by RIS files and (3) efficient processing techniques for relational operations using RIS files and functional dependencies.

RIS files are actually implemented as the access path structure of a research information system ARIS, which have been developed at Yajima Laboratory of Kyoto University [YAJIK8001] [TANAY8003] [LEVII8003]. Some experimental results are shown in

[LEVII8003]. Query processing strategies using RIS files are also shown in [TANAY8003]. Further applications of RIS files are discussed in [LEVIK7911].

In RIS files, extra storage space, when compared with ordinary inverted-file based database systems, will be required for the hashed values. We have not evaluated formally this size problem. However, the following should be noted. (1) Our policy is to provide a partial inversion system by using RIS files. We don't intend to create RIS files for all attributes. (2) A RIS file combines multiple ordinary secondary indexes for domain-related attributes. Therefore, we can eliminate duplicates of common domain values appearing in those multiple indexes. The size of a RIS file without the hashed values will be smaller than the size of multiple secondary indexes.

As a future problem, it is important to consider how to select the attributes for which RIS files are constructed and how to select attributes whose values are hashed. This problem is concerned with selecting an optimum search strategy using multiple RIS files.

In order to achieve a good performance in a relational database system, two approaches are considered: One is to implement a system supported by an efficient index organization. The other is a database machine approach. Even in a database machine approach, the importance of an index file organization has been recognized in order to process join operations efficiently, for example, DBC (Data Base Computer) in [BANEH7906]. RIS file organization will be also available in database machines as an index organization.

CHAPTER 8 CONCLUSION

In the design and implementation of database systems, retrieval time, required storage space and the ease of update operations are the important criteria of the performance. There generally exists a trade-off among these three criteria, and therefore, it is important to consider the logical design and the physical design of a database depending upon which criterion is especially important in the database system. The logical design of a database offers a logical view of data to users. It is important to design a logical view of data that makes it easy to maintain the consistency of data against update operations, and that reduces the logical redundancy of data. The physical design of a database is substantial when we wish to reduce the required storage space and/or to achieve a rapid retrieval and a rapid update.

In this thesis, some important problems in the logical design and the physical design of a relational database are identified, and several new results are obtained. We have discussed five major topics: (1) properties of embedded multivalued dependencies (EMVDs), (2) preservation of data dependencies (3) semantic aspects of data dependencies, (4) organization of Quasi-Consecutive Retrieval (QCR) files and (5) organization of Relational Inverted Structure (RIS) files. The first three topics are concerned with the logical design of a relational database, especially with the specification and maintenance of an important class of integrity constraints about attribute relationships, called data dependencies. The last topics are concerned with the physical data organization of a relational database system.

From Chapter 3 to Chapter 5, several important problems of

data dependencies, which play a central role in the logical design theory of a relational database, are investigated. The major results are as follows:

(a) Some new inference rules for EMVDs are given. A necessary and sufficient condition for an MVD (multivalued dependency), which holds in a certain set of attributes, to hold in its superset of attributes is derived from the inference rules for EMVDs. Furthermore, some useful conditions to reduce the total number of attributes of a given set of EMVDs is also shown.

(b) A class of data dependencies that can be preserved totally by a set of relation schemes is investigated. We allow each relation on any relation scheme to be updated independently from others. A necessary and sufficient condition for a set of relation schemes to preserve a data dependency (functional, embedded multivalued or embedded join dependency) is provided when each relation scheme is in 'Dependency Preserving Normal Form (DPNF)'.

(c) Semantic analysis of data dependencies, especially functional and multivalued dependencies, is provided. Several disagreements between these data dependencies and user's conceptual information structure are shown. Especially, some transitively specified multivalued dependencies are shown to often cause semantically 'unnatural' constraint on data. Some condition to cope with this problem is also provided.

The result (a) is obtained from the following motivations: One is the difficulties of correct specifications of MVDs because the validity of an MVD depends on the underlying set of attributes of a relation scheme. The other is that any theory has not been known to analyze what MVDs are represented by Fagin's decomposition approach. Since both of these problems are strongly related to EMVDs, we have studied properties of

EMVDs. Although it was recently shown that there does not exist a general finite complete set of inference rules for EMVDs, the following problems will be necessary to be investigated: To find an efficient algorithm which computes all the EMVDs implied by given EMVDs of a 'fixed' relation scheme, and to find an algorithm to transform given EMVDs into equivalent EMVDs with the total number of attributes minimized.

The major difference of the result (b) from others is that we allow each relation to be updated independently from others. That is, any update operation to a relation is allowed if and only if the updated result also satisfies the given data dependencies on the relation scheme. The notion of DPNF relation schemes are introduced, but the assumption that each relation scheme is in DPNF is not so strict. Because any relation scheme in BCNF or Fagin's 4NF is always in DPNF. As a future problem, it will be necessary to consider an efficient algorithm to test whether a data dependency is preserved by a set of not necessarily DPNF relation schemes.

The result (c) is useful because of the following reasons: One is that there has not been sufficient analysis of the semantics of data dependencies from the viewpoint of 'real world' description. The other is that the disagreement between MVDs and user's conceptual information structure is shown to be larger than expected. As a future problem, it will be necessary to construct a design method which automatically transforms a given conceptual design into a set of correct data dependencies.

The physical data organizations for a relational database are investigated in Chapter 6 and Chapter 7. Two file organization methods are introduced, which are useful to reduce the redundancy of storage space for relations and secondary indices, and also useful to achieve a rapid retrieval. The

major results are as follows:

(d) The notion of QCR files, which is a generalization of Ghosh's concecutive retrieval file by the concept of 'buffer size', is introduced. A necessary and sufficient condition that a QCR file without any redundancy can be constructed is given from the viewpoint of graph theory. Furthermore, a heuristic algorithm for organizing a QCR file with less redundancy is also shown.

(e) A new index organization called RIS and the concept of 'pseudo operations' by hashed values are introduced. By using RIS files and the pseudo operations, we provide an efficient query processing method for relational database operations.

The QCR file organization is useful to store secondary indices with less redundancy. It is also useful to reduce the total number of accesses to relations when several relations are also organized as QCR files. The disadvantage of QCR files is that if some data is updated, it will be necessary to reorganize some QCR files. To find an efficient reorganization algorithm and to select which data should be organized as QCR files are future problems. As for RIS file organization, it is a future problem to evaluate precisely the performance compared with conventional secondary indexes or links.

Acknowledgements

I would like to express my sincere appreciation to Professor Shuzo Yajima of Kyoto University for his suggestions, accurate criticisms and continuous encouragements during this research.

I would also like to thank Associate Professor Yahiko Kambayashi of Kyoto University for his continuous guidance, encouragements and invaluable comments through the work on this thesis.

I should thank Professor Masaaki Kawaguchi of Kobe University for his kind encouragements. Thanks are due to Dr. Kosaku Inagaki of Kyoto University for his valuable comments on the problem in Chapter 6. Thanks are also due to Dr. Syunsuke Uemura of the Electrotechnical Laboratory for presenting me a lot of papers on databases.

Additionally, thanks are due to the members of Professor Yajima's research laboratory for their kind discussions and cooperations. Especially, Mr. Hiroto Yasuura kindly offered me a text formatting computer program, by which most of this thesis was printed. Mr. Le Viet Chung kindly gave me valuable comments on this research. Furthermore, the implementation of RIS files in Chapter 7 was performed by his cooperations.

This research was partly supported by Sakko-kai Foundation.

References

[AHO-B7909]

Aho,A.V., Beeri,C. and Ullman,J.D., 'The Theory of Joins in Relational Databases', ACMTODS, Vol.4, No.3, pp.297-314, September 1979.

[ANSI7502]

ANSI/X3/SPARC, 'Interim Report', FDT, Quarterly Bulletin of ACM-SIGMOD, Vol.7, No.2, February 1975.

[ARMS7408]

Armstrong,W.W., 'Dependency Structures of Database Relationships', Information Processing 74, North-Holland, Amsterdam, pp.580-583, August 1974.

[ASTRB7606]

Astrahan et al., 'System R: Relational Approach to Database Management', ACMTODS, Vol.1, No.2, June 1976.

[BANEH7906]

Banerjee,J., Hsiao,D.K. and Kannan,K., 'DBC - A Database Computer for Very Large Databases', IEEE Trans. on Computer, Vol.C-28, No.6, pp.414-429, June 1979.

[BEER7901]

Beeri,C., 'On the Role of Data Dependencies in the Construction of Relational Database Schemas', Dept. of Computer Science, The Hebrew University, Report No.43, January 1979.

[BEERB7809]

Beeri,C., Bernstein,P.A. and Goodman,N., 'A Sophisticate's Introduction to Database Normalization Theory', Proc. 4th International Conference on VLDB, pp.113-124, September 1978.

[BEERF7708]

Beeri,C., Fagin,R. and Howard,J., 'A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations',

Proc. 1977 ACM SIGMOD International Conference of Management of Data, pp.47-61, August 1977.

[BEERM7904]

Beeri,C., Mendelzon,A.O., Sagiv,Y. and Ullman,J.D., 'Equivalence of Relational Database Schemes', Proc. 11th Annual ACM Symposium on Theory of Computing, pp.319-329, April 1979.

[BEERV7911]

Beeri,C. and Vardi,M.Y., 'On the Properties of Total Join Dependencies', Dept. of Computer Science, The Hebrew University of erusalem, Research Report, November 1979.

[BEERR8001]

Beeri,C. and Rissanen,J., 'Faithful Representations of Relational Database Schemes', IBM Research Report, No.RJ2722, January 1980.

[BERN7612]

Bernstein,P.A., 'Synthesizing Third Normal Form Relations from Functional Dependencies', ACMTODS, Vol.1, No.4, pp.277-298, December 1976.

[BISK78]

Biskup,J., 'On the Complementation Rule for Multivalued Dependencies in Database Relations', Acta Informatica, Vol.10, Fasc.3, 1978.

[BISK8001]

Biskup,J., 'Inferences of Multivalued Dependencies in Fixed and Undetermined Universes', Theoretical Computer Science, Vol.10, No.1, January 1980.

[BOOTL7505]

Booth,K.S. and Lueker,G.S., 'Linear Algorithm to recognize interval graphs and Test for the Consecutive Ones Property', Proc. of 7th Annual ACM Symp. on Theory of Computing, pp.255-265, May 1975.

[BOOTL7612]
Booth,K.S. and Lueker,G.S., 'Testing for the Consecutive Ones Property, Interval graphs, and Graph Planarity Using PQ-tree Algorithms', J. Comp. System Sci., Vol.13, No.3, pp.335-379, December 1976.
[BREU73]
Breuer,M.A. (Ed.), 'Design Automation of Digital Systems', Vol.1. Theory and Techniques', Prentice-Hall, Englewood Cliffs, New Jersey, 1973.
[CHEN7603]
Chen,P.P.-S., 'The Entity-Relationship Model - Toward a Unified View of Data', ACMTODS, Vol.21, No.1, March 1976.
[CHAM7602]
Chamberlin,D.D., 'Relational Data Base Management Systems', IBM Research Report, RJ1729, February 1976.
[CODD7006]
Codd,E.F., 'A Relational Model of Data for Large Shared Data Banks', Comm. ACM, Vol.13, No.6, pp.377-387, June 1970.
[CODD7105F]
Codd,E.F., 'Further Normalization of the Data Base Relational Model', Proc. Courant Computer Science Symposium 6, Data Base Systems, pp.33-64, May 1971.
[CODD7105R]
Codd,E.F., 'Relational Completeness of Data Base Sublanguages', Proc. Courant Computer Science Symposium 6, Data Base Systems, pp.65-98, May 1971.
[CODD7912]
Codd,E.F., 'Extending the Database Relational Model to Capture More meaning', ACMTODS, Vol.4, No.4, pp.397-434, December 1979.
[DATE77]
Date,C.J., 'An Introduction to Database Systems', 2nd Edition,

Addison-Wesley, 1977.

[ESWA7503]

Eswaran,K.P., 'Faithful Representation of a Family of Sets by a Set of Intervals', SIAM J. Comp., Vol.4, No.1, pp.56-68, March 1975.

[FAGI7709]

Fagin,R., 'Multivalued Dependencies and a New Normal Form for Relational Databases', ACMTODS, Vol.2, No.3, pp.262-278, September 1977.

[FAGI7710]

Fagin,R., 'The Decomposition versus the Synthetic Approach to Relational Database Design', Proc. 3rd International Conference on VLDB, pp.441-446, October 1977.

[FAGI8003]

Fagin,R., 'Horn Clauses and Database Dependencies', IBM Research Report, No.RJ2741, March 1980.

[FULKG65]

Fulkerson,D.R. and Gross,O.A., 'Incidence Matrices and Interval Graphs', Pacific J. Math., Vol.15, No.3, pp.835-855, 1965.

[GHOS7209]

Ghosh,S.P., 'File Organization: The Consecutive Retrieval Property', Commun. ACM, Vol.15, No.9, pp.802-808, September 1972.

[GHOS7508]

Ghosh,S.P., 'The Consecutive Storage of Relevant Records with Redundancy', Commun. ACM, Vol.18, No.8, pp.464-470, August 1975.

[GHOS76]

Ghosh,S.P., 'Data Base Organization for Data Management', Academic Press, New York, 1976.

[GILMH64]

Gilmore,P.C. and Hoffman,A.J., 'A Characterization of

Comparability Graph and of Interval Graphs', Can. J. Math., No.16, pp.539-548, 1964.

[HAER7610]

Haerder,T., 'An Implementation Technique for a Generalized Access Path Structure', IBM Research Report, October 1976.

[HALL07601]

Hall,P.A.V., Owlett,J. and Todd,S.J.P., 'Relations and Entities', Proc. IFIP TC-2 Working Conference on Modelling in Data Base Management Systems, pp.201-220, January 1976.

[HONEL8002]

Honeyman,P., Ladner,R.E. and Yannakakis,M., 'Testing the Universal Instance Assumption', Information Processing Letters, Vol.10, No.1, pp.14-19, February 1980.

[ITO-T8009]

Ito,M., Taniguchi,K. and Kasami,T., 'Inference of Embedded Multivalued Dependencies in Relational Databases' (in Japanese), Trans. IECEJ, Vol.J63-D, No.9, pp.683-690, September 1980.

[KAMB7906]

Kambayashi,Y., 'A New Synthetic Approach for Relational Database Design', presented at AFIPS NCC, June 1979.

[KAMB81]

Kambayashi,Y. (Ed.), 'Database : A Bibliography', to be published by Computer Science Press.

[KATS8003]

Katsuno,H., 'On the Semantics of Multivalued Dependencies' (in Japanese), Paper of Technical Group on Automata and Languages of IECEJ, TGAL79-115, March 1980.

[MAIEM7912]

Maier,M., Mendelzon,A.O., Sadri,F. and Ullman,J.D., 'Adequacy of Decompositions of Relational Databases', Proc. Workshop on Formal Bases for Data Bases, Toulouse, December 1979.

[MAKI7710]
Makinouchi,A., 'A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Database Model', Proc. 3rd International Conference on VLDB, pp.447-453, October 1977.
[MEND7901]
Mendelzon,A.O., 'On Axiomatizing Multivalued Dependencies in Relational Databases', JACM, Vol.26, No.1, pp.37-44, January 1979.
[MENDM7910]
Mendelzon,A.O. and Maier,D., 'Generalized Mutual Dependencies and the Decomposition of Database Relations', Proc. 5th International Conference on VLDB, pp.75-82, October 1979.
[NAKAC8011]
Nakamura,C. and Chen,P.P., 'A Consideration on the Transitivity Problem of Multivalued Dependencies in Relational Databases' (in Japanese), Paper of Technical Group on Data Base Management Systems of IPSJ, November 1980.
[NICO7805]
Nicolas,J.M., 'First Order Logic Formalization for Functional, Multivalued and Mutual dependencies', Proc. ACM-SIGMOD International Conference, pp.40-46, May 1978.
[PARKP8005]
Parker,D.S.Jr. and Parsaye-Ghomi,K., 'Inferences Involving Embedded Multivalued Dependencies and Transitive Dependencies', Proc. ACM SIGMOD International Conference on Managemant of Data, May 1980.
[PNUEL71]
Pnueli,A., Lempel,A. and Even,S., 'Transitive Orientation of Graphs and Identification of Permutation Graphs', Can. J. Math., Vol.23, No.1, pp.160-175, 1971.

[RISS7712]
Rissanen,J., 'Independent Components of Relations', ACMTODS, Vol.2, No.4, pp.317-325, December 1977.
[RISS7809]
Rissanen,J., 'Theory of Relations for Databases - A Tutorial Survey', Proc. 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science 64, pp.536-551, September 1978.
[SAGIW7907]
Sagiv,Y. and Walecka,S., 'Subset Dependencies as an Alternative to Embedded Multivalued Dependencies', Dept. of Computer Science, University of Illinois at Urbana-Champaign, Report UIUCDCS-R-79-980, July 1979.
[SCHM7701]
Schmid,H.A., 'An Analysis of Some Constructs for Conceptual Models', Proc. IFIP TC-2 Working Conference on Architecture and Models in Data Base Management Systems, pp.119-148, January 1977.
[SCHMS7505]
Schmid,H.A. and Swenson,J.R., 'On the Semantics of the Relational Data Model', Proc. ACM-SIGMOD International Conference, pp.211-223, May 1975.
[ULLM80]
Ullman,J.D., 'Principles of Database Systems', Computer Science Press, Inc., 1980.
[WAKSG7402]
Waksman,A. and Green,M.W., 'On the Conscutive Retrieval Property in File Organization', IEEE Trans. Comp., Vol.c-27, No.2, pp.173-174, February 1974.
[YAMAU77]
Yamamoto,S., Ushio,K., Tazawa,S., Ikeda,H., Tamari,F. and

Hamada,N., 'Partition of a Query Set into Minimal Number of Subsets Having Consecutive Retrieval Property', J. Statistical Planning and Inference, Vol.1, No.1, 1977.

[ZANI7607]

Zaniolo,C, 'Analysis and Design of Relational Schemata for Database Systems', Computer Methodology Group Report, UCLA-ENG-7669, Dept. of Computer Science, University of California at Los Angels, July 1976.

IECEJ: The Institute of Electronics and Communication Engineers of Japan.

IPSJ: Information Processing Society of Japan.

LIST OF MAJOR PUBLICATIONS

[KAMBT7710]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'A Relational Data Language with Simplified Binary Relation Handling Capability', Proc. 3rd International Conference on Very Large Data Bases, pp.338-350, October 1977.

[TANAL7808]
Tanaka,K., Le Viet,C., Kambayashi,Y. and Yajima,S., 'A File Organization Suitable for Relational Database Operations', Proc. International Conference on Mathematical Studies of Information Processing, pp.183-214, August 1978. Also appeared in Lecture Notes in Computer Science 75, Springer-Verlag, pp.193-224, 1979.

[TANAK79]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Organization of Quasi-Consecutive Retrieval Files', Information Systems, Vol.4, No.1, pp.23-33, 1979.

[TANAK7904]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Representability Problem for Relational Database Design by Multivalued Dependencies', Memoirs of the Research Institute for Mathematical Sciences, Kyoto University, 353, pp.1-10, April 1979.

[LEVIK7904]
Le Viet,C., Kambayashi,Y., Tanaka,K. and Yajima,S., 'Query Processing in a Relational Database Using Abstracted Characteristics of Data', Memoirs of the Research Institute for Mathematical Sciences, Kyoto University, 353, pp.20-29, April 1979.

[TANAK7908]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Properties of Embedded

Multivalued Dependencies in Relational Databases', Transactions of the Institute of Electronics and Communication Engineers of Japan (IECEJ), Vol.E62, No.8, pp.536-543, August 1979.

[KAMBT7911S]

Kambayashi,Y., Tanaka,K. and Yajima,S., 'Semantic Aspects of Data Dependencies and Their Application to Relational Database Design', Proc. 3rd International Computer Software and Applications Conference (COMPSAC), pp.398-403, November 1979.

[LEVIK7911]

Le Viet,C., Kambayashi,Y., Tanaka,K. and Yajima,S., 'Use of Abstracted Characteristics of Data in Relational Databases', Proc. 3rd International Computer Software and Applications Conference (COMPSAC), pp.409-414, November 1979.

[KAMBT7911R]

Kambayashi,Y., Tanaka,K. and Yajima,S., 'Relational Database Design', Proc. 13th Japan IBM Computer Science Symposium (Working Conference on Database Engineering), pp.I-1 - I-36, November 1979.

[YAJIK8001]

Yajima,S., Kambayashi,Y., Konishi,O., Tanaka,K., Le Viet,C. and Kato,T., 'Bibliographic Information Processing Facilities for Relational Database System ARIS', Proc. 13th Hawaii International Conference on System Sciences, Vol.II, pp.198-207, January 1980.

[TANAK8002]

Tanaka,K., Kambayashi,Y. and Yajima,S., 'Preservability of Data Dependencies for Relational Database Operations', Proc. JIPDEC Information Systems Seminar on Semantic Aspects of Databases, pp.151-174, February 1980.

[LEVIK8002]

Le Viet,C., Kambayashi,Y., Tanaka,K. and Yajima,S.,

'Implementation and Evaluation of Relational Inverted Structure (RIS) Files', Proc. JIPDEC Information Systems Seminar on Semantic Aspects of Databases, pp.194-242, February 1980.

[KAMBT81]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'Problems of Relational Database Design', to appear in Data Base Engineering, Lecture Notes in Computer Science, 1981.

[KAMB81]
Kambayashi,Y. ed., (Konishi,O., Tanaka,K. and Le Viet,C.: Editorial Assistants), 'Database: A Bibliography', to be published by Computer Science Press, 1981.

JIPDEC: Japan Information Processing Development Center.

LIST OF TECHNICAL REPORTS

[TANAK7703]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Organization of Quasi-Consecutive Retrieval Files without Duplicates of Records' (in Japanese), Paper of Technical Group on Data Bases of IPSJ, DB33-2, March 1977.

[KAMBT7709]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'A Relational Data Language with Simplified Binary Relation Handling Capability and Related Subjects', Paper of Technical Group on Automata and Languages of IECEJ, TGAL77-38, September 1977.

[KAMBT7801]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'Storage Anomalies and Normalization Problems in Relational Databases' (in Japanese), Paper of Technical group on Automata and Languages of IECEJ, TGAL77-71, January 1978.

[KAMBT7805]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'A Conceptual-level Schema for Relational Databases' (in Japanese), Paper of Technical Group on Automata and Languages of IECEJ, TGAL78-11, May 1978.

[TANAK7901]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Properties of Embedded Multivalued Dependencies and Their Application', Paper of Technical Group on Automata and Languages of IECEJ, TGAL78-84, January 1979.

[TANAY8003]
Tanaka,K., Yamada,M., Kambayashi,Y., Yajima,S. and Yamamura,M., 'Design and Implementation of Query Processing Subsystem in the Relational Database System ARIS' (in Japanese), Paper of

Technical Group on Electric Computers of IECEJ, TGEC79-86, March 1980.

[LEVII8003]

Le Viet,C., Imai,Y., Tanaka,K., Kambayashi,Y. and Yajima,S., 'Design and Implementation of a Data Storage and Retrieval Subsystem Using Relational Inverted Structure (RIS) Files' (in Japanese), Paper of Technical Group on Electric Computers of IECEJ, TGEC79-87. March 1980.

IECEJ: The Institute of Electronics and Communication Engineers of Japan.

IPSJ: Information Processing Society of Japan.

LIST OF CONVENTION RECORDS

[TANAK7510]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'Organization of Quasi-Consecutive Retrieval Files' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G8-13, October 1975.
[KAMBT7603]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'An Organization Algorithm for Quasi-Consecutive Retrieval Files' (in Japanese), National Convention Record of IECEJ, 1116, March 1976.
[TANAK7611]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'An Organization Method for Quasi-Consecutive Retrieval Files with Reduced Redundancy' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G7-24, November 1976.
[TANAK7710]
Tanaka,K., Kambayashi,Y. and Yajima,S., 'A Relational Data Language with Simplified Binary Relation Handling Capability' (in Japanese), National Convention Record of IPSJ, 42, October 1977.
[KAMBT7710P]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'Properties of Compound Values and Their Application to the Representation of Data Dependencies in Relational Databases' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G6-17, October 1977.
[KAMBT7710H]
Kambayashi,Y., Tanaka,K. and Yajima,S., 'A Hierarchy of Complexities of Data Dependencies in Relational Databases' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G6-18, October 1977.
[TANAL7710]

Tanaka,K., Le Viet,C., Kambayashi,Y. and Yajima,S., 'Update Viability through Views in Relational Databases' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G6-19, October 1977.

[TANAK7803]

Tanaka,K., Kambayashi,Y. and Yajima,S., 'Problems of Several Kinds of Data Dependencies in Relational Databases' (in Japanese), National Convention Record of IECEJ, 1170, March 1978.

[TANAL7808I]

Tanaka,K., Le Viet,C., Kambayashi,Y. and Yajima,S., 'A Relational Inverted Structure (RIS) Files for Database Operations', National Convention Record of IPSJ, 1A-2, August 1978.

[TANAK7808]

Tanaka,K., Kambayashi,Y. and Yajima,S., 'Properties of Embedded Multivalued Dependencies in Relational Databases' (in Japanese), National Convention Record of IPSJ, 1A-5, August 1978.

[KAMBT7808]

Kambayashi,Y., Tanaka,K. and Yajima,S., 'Representation of a Conceptual Model in Relational Databases' (in Japanese), National Convention Record of IPSJ, 1A-6, August 1978.

[KAMBT7810]

Kambayashi,Y., Tanaka,K., Imai,Y. and Yajima,S., 'Generalized Null Values in Relational Databases and Their Application to Database Decompositions' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G6-13, October 1978.

[TANAL7810]

Tanaka,K., Le Viet,C., Kambayashi,Y. and Yajima,S., 'A Laboratory Information System ARIS Based on the Relational Data Model' (in Japanese), Kansai Branch Joint Convention Record of

EERSJ, G6-14, October 1978.

[TANAY7810]

Tanaka,K., Yamamura,M., Kambayashi,Y. and Yajima,S., 'User Language Processing Subsystem in Laboratory Information System ARIS' (in Japanese), Kansai Branch Joint Convention Record of EERSJ, G6-15, October 1978.

[LEVIT7810]

Le Viet,C., Tanaka,K., Kambayashi,Y. and Yajima,S., 'Relational Data Storage and Access System for Laboratory Information System ARIS', Kansai branch Joint Convention Record of EERSJ, G6-16, October 1978.

[TANAK7907]

Tanaka,K., Kambayashi,Y. and Yajima,S., 'Design Criteria for Relational Databases by Considering Multivalued Dependencies' (in Japanese), National Convention Record of IPSJ, 1D-4, July 1979.

[TANAY7911]

Tanaka,K., Yamada,M., Kambayashi,Y. and Yajima,S., 'Testing Validity of Queries in Relational Databases' (in Japanese). Kansai Branch Joint Convention Record of EERSJ, G8-25, November 1979.

[TANAK8005]

Tanaka,K., Kambayashi,Y. and Yajima,S., 'Preservations of Data Dependencies for Joins in Relational Databases' (in Japanese), National Convention Record of IPSJ, 1G-1, May 1980.

EERSJ: Electric, Electronics and Related Societies of Japan.